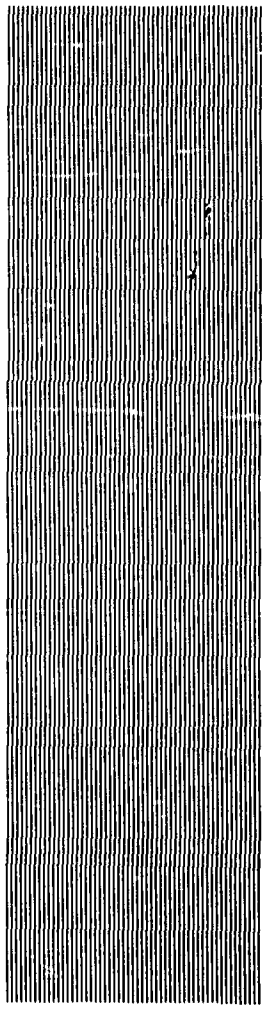
UNLIMITED

Report No. 89014

ROYAL SIGNALS AND RADAR ESTABLISHMENT, MALVERN

THE SMITE APPROACH TO SECURITY

Author: P F Terry



PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE
RSRE
Malvern, Worcestershire.

August 1989

UNLIMITED

CONDITIONS OF RELEASE

0051796 BR-111536

********** U

********** Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

ROYAL SIGNALS AND RADAR ESTABLISMENT

Report 89014

Title: The SMITE Approach to Security

Author: P. F. Terry[†]

Date: August 1989

ABSTRACT

SMITE is a novel computer architecture implementing a new security policy model. It is proposed as the best available technology which may be used to develop information systems for operational use where high assurance of complex confidentialty and integrity based security policies is required.

This report records the results of work carried out by Retix Systems for CC1 division RSRE. This contract provided a peer review of their architecture proposals, characterised the essential architectural elements and formulated the security oriented top level model. In this way it provided a baseline definition of SMITE to aid the future way forward for the project.

This report has been furnished for use by Her Majesty's Government under the terms and conditions of contract A94c/2711 from the Royal Signals and Radar Establishment.

[†] Retix Systems Ltd
20 Alan Turing Road
Surrey Research Park
Guildford, Surrey, GU2 5YF

CONTENTS

1. INTRODUCTION

2. OVERVIEW

3. SECURITY POLICY MODEL

4. THE SMITE ARCHITECTURE

References

1. INTRODUCTION

Project Background

SMITE has resulted from a programme of research by RSRE into Multi-Level Computer Security the origins of which can be traced to work begun over a decade ago. At an early stage of this effort RSRE issued a contract for a study to produce recommendations for the best use of effort towards widening the knowledge of the issues and requirements for computer security in the UK. This contract, awarded to Plessey, resulted in a comprehensive report on computer security in computer networks [Andrews81].

The Plessey report recognised that the then current RSRE projects formed a low-level application of existing techniques to existing problems and thus formed an approach which we might with hindsight characterise as a "nuts and bolts" approach. As a way forward the report recommended the immediate deployment of effort to produce a "mid-range" Trusted Computing Base (TCB) system for limited functionality network devices that would be required in the near future. In the background a longer term research effort should be mounted to produce highly secure general purpose computer systems.

The report identified three phases for this Secure Communications Processor research programme [Barnes85]. SCP1 was a collective term for the then extant approaches, SCP2 was the development of the mid-range TCB based on current hardware and software developments, and finally SCP3, the general purpose computer system, for which specialised hardware and software techniques would be applied.

SMITE is in fact the name of the third phase, SCP3, project [Wiseman86a, Wiseman86b]. The name has been changed in recognition of the fact that SMITE is a general purpose computer system and is not limited to network component functionality as the term SCP3 might imply.

Technical Background

The SMITE proposals for secure system development have been influenced from a number of sources which will be referenced throughout this document.

Our modelling approach is basically that expounded in the seminal "Mathematical Foundations" paper of Bell-LaPadula [Bell73a] in that it uses state machine concepts, however our model, the notions of confidentiality, the axioms, and techniques used are profoundly different form those found in the later papers [Bell73b,Bell74 and Bell76]. Our approach has been influenced from a number of sources such as Bell-LaPadula, McLean [McLean87] and various information flow techniques such as Non-Interference [Goguen82] and Separability [Rushby81].

Our approach to policy issues has been profoundly influenced by the many useful discussions and workshops which arose from the Clark-Wilson paper, "A Comparison of Commercial and Military Security Policies" [Clark87]. Briefly, the concept of the requirement for consistency between the internal system state and the external environment, presented in their model as the Separation of Duty concept, is taken to be the fundamental security mechanism of not only integrity but confidentiality and assurance in general.

The architecture and software development approach proposed is a direct development of work by RSRE's CC2 division. [Foster82a, Foster82b, Foster83, Currie82, Currie85, Foster89]

The existence of a team with the experience and inspiration to draw these many disparate elements together into the SMITE proposals is entirely due to the foresight and perseverance of RSRE's CC1 division in maintaining the SCP research programme, which has produced convincing demonstrations of research results such as SCP2 and DSS.

History of this Document

This report is an improved presentation of the work reported in the Retix Systems Final Technical Report [Terry89] and the 1989 Security and Privacy Symposium paper [Terry-Wiseman89].

The symposium paper was generated against tight deadlines to report the very latest developments in the SMITE approach to policy modelling. Because of this the formal description of the model was not as carefully peer-reviewed as would normally be desirable and contains a number of detailed, technical flaws and one or two editorial mistakes.

Most of these resulted from attempts to address some of the concerns of the paper's referees by the introduction of some "stylistic improvements" which on closer examination are wrong. Having met the paper submission deadlines the problem has been compounded by the fact that the technical report, upon which the paper was originally based, was simply brought into line with the erroneous paper in the closing stages of the contract.

This report superceeds both the above references and has been produced after sober reflection of the previous formal models and is now believed to be definitive.

Structure of this Document

Section 2, Overview, is a fairly expansive, informal presentation of the overall SMITE approach to security. It develops concepts of security from scratch and relates these concepts to existing work and practice in an entirely informal manner.

Section 3, Security Policy Model, presents the formal policy model and axioms which capture the concepts developed in the Overview. The basic execution nature of the model is shown to consist of a small number of basic transitions which can be embodied in a small number of transition rules. Although the model elements and transitions are quite simple it is shown that it is sufficient to model complex application and general purpose computer systems. Finally this section proves that the axioms of the model capture the confidentiality notions of security by means of a non-interference proof of the model.

Section 4, The SMITE Architecture, describes the underlying architectural features which are required to implement the policy model. It then goes on to describe the intended architecture for use in the SMITE project and describes the correspondence between the architecture and the policy model in an informal manner. In an operational development the correspondence would be formally demonstrated by continuing the refinement of the model to design, to implementation.

2. OVERVIEW

SMITE is an approach to producing secure information systems. This leads us to ask what is security? A standard dictionary definition is as follows.

> **security** n. **1** the state of being secure. **2** assured freedom from poverty or want.... Archaic carelessness or overconfidence
>
> **secure** adj. **1** free from danger, damage etc. **2** free from fear, care etc..... Archaic careless or overconfident. [From Latin securus free from care, from se- without + cura, care]

This gives us the typical manufacturers view of a customer definition of security, "I want a system that I don't care about because I am assured that it is free from all the etc's which might assail me in the future but which I can't enumerate at the moment". The archaic definitions of carelessness and overconfidence seem to shout out from the past the dire consequences of such an approach and echo the concerns of the modern prophets of doom, "No security is better than illusory security".

A sophisticated view of security in these terms can however be described as assurance of freedom from fear of specified attacks against specified elements of a system. To define security in these terms is a tripartite affair,

i. specify which properties of which elements of the system are important,

ii. specify what protection from which attacks a threat analysis requires,

iii. specify how much assurance is required that such defences are successful.

Taken in its general English sense this definition of security is not at odds with either military or commercial security practice.

Military Security

The military and government arena has in the past taken a more definitive approach to security and attempted to codify and lay down standards for secure systems. An impartial view of these attempts shows that quite naturally they have emphasised a particular property, confidentiality, as paramount to security.

In codifying these concepts the military have produced many formal definitions, or models, of confidentiality. A dictionary analysis of this term will suffice for our discussions here and is as follows.

Confidential adj. **1** spoken,written, or given in confidence; secret; private. **2** entrusted with anothers confidence or secret affairs.

Confidence n. **1** feeling of trust in a person or thing. ..... **4** something confided or entrusted. **5 in confidence** as a secret.

Secret adj. **1** kept hidden or separate from the knowledge of others. **4** able or tending to keep things private or to oneself.

private adj. **1** not widely known; confidential; secret. [from Latin privatus belonging to one individual]

Confidentiality from the above seems to be about knowledge of things and its distribution amongst individuals.

The aspect of confidentiality that things are kept secret or private is an easily stated and implemented requirement for the discrete identification of things and ensuring that they are attributed to only one individual, an isolation policy.

> **policy** n.**1** a plan of action adopted or pursued by an individual, government, party, business, etc.

The assurance of such a policy can be very high because it is basically saying that if you want something done in such a way that no one else knows about it - do it yourself.

The notion that distribution of things amongst individuals is allowed "in confidence" in the definition recognises that in reality this isolation policy is not practical and that one is forced to delegate tasks. This problem of delegating or sharing a task with others leads to a quite natural extension of isolation policy as follows. If things are isolated and are secret, that is private, known only to one individual, then if an individual can establish a basis of trust in another individual he may pass knowledge of a secret thing to that other individual.

In this light a generic confidentiality policy requires that things are isolated and attributable to individuals, lays down the means by which a basis of trust in individuals is established, ie that they don't pass on or leak secrets given to them, and requires that things are not delegated to multiple individuals other than as permitted by properly established trust.

Commercial Security

Some have argued that impartial analysis of commerical security shows it to be biased to a particular property, which is universally referred to as "integrity".

integrity n. **1** adherence to moral principles; honesty. **2** the quality of being unimpaired; soundness. **3** unity; wholeness(see INTEGER).

honest adj. **1** not given to lying, cheating, stealing etc.; trustworthy.

sound^{2} adj. **1** free from damage, injury, decay etc..... **5** valid, logical or justifiable.

This word and its definition is fraught with overloaded technical definitions and empirical uses and at this point we unavoidable run up against the problem of pre-conceived notions and ideas of "integrity". In order to "smooth the way" for the later expositions of the concept in our model we will digress for a moment here and discuss these notions.

We believe that the security community has, and confuses, two interpretations of the term integrity which correspond to the first and second facets of the dictionary definition. The dichotomy of abuse and confusion which arises from this fact can be seen by following the example of the classification and downgrading of documents.

> "Obviously, a system of controls on the handling of documents which is driven by the labels on those documents is dependent on the integrity of those labels".
>
> "When a document is downgraded the process responsible for that action must function with high integrity to prevent mishap".

In the former use the emphasis is clearly on simple correctness.

> **correct** ...adj. **6.** free from error; true; accurate

When documents are merged or quoted the label on the new document must be correctly computed from the original labels. This is sufficient in this context because even if it is not the done thing to put the war plans in an office party invitation the integrity of the resultant label will ensure the confidentiality of the war plans in that most invitations will be prevented from distribution to their intended recipients.

In the latter use it is clearly not sufficient and connotations of "the done thing" are clearly required. The weeding and downgrading process of the "30 year rule" is clearly correct to change any classification label to unclassified but only for appropriate documents, more than 30 year old cabinet papers and not 20 year old papers or seventy year old papers relating to the monarch which should be subject to the 100 year rule.

To distinguish these uses we reserve the term integrity for the former, correctness oriented, aspects and use "appropriateness" for the latter.

> **appropriate** adj. **1.** right or suitable;fitting. [...see PROPER]
>
> ***proper*** *adj.* **1.** *appropriate or suited for some purpose.*
> **2.** correct in behaviour or conduct.
> **3.** excessively correct in conduct; vigorously moral

Thus integrity is taken by us to imply connotations of the property of a thing or state; that it is correct. Appropriateness implies connotations of the property of active behaviour, transitions or functions etc; that it is the right thing to do. The contrast between these two notions in terms of state property versus a transition property is very instructive especially in the context of disentangling confusion which arises over "integrity" models such as [Biba77], [Shockley87] and [Lee88].

In a state machine environment there is an inevitable interdependence of state properties on transition properties. Thus if we consider the integrity, in the sense of correctness, of a state, for example all objects are labelled, there is a consequent requirement for a property of transitions that they uphold the state integrity in the the subsequent state, for example when an object is created it is indeed labelled. This transition property is not the set of properties which we regard as being the "appropriateness" properties of the transition, they are merely a low-level adjunct to the basic state property.

Continuing the example, what we would regard as an "appropriateness" property would be the fact that the particular label for the new object is the "proper" one given the circumstances of the objects creation. Thus an untrusted user creating the object would require that the label of the new object dominates any information potentially incorporated in it by the user, as reflected by the users current security level, whereas a trusted subject may have free reign (in typical Bell-LaPadula parlance).

We believe that the integrity models cited above are essentially integrity in the sense of preservation of the correctness of the state which can be extended to provide strictly limited control in the sense of restricting users to "appropriate" actions. They are limited in the sense that the "appropriateness" must be expressible in simple state relationships such as the object creation example above. This can be seen in the way that both [Shockley87] and [Lee88] (which are applications of the category aspects of [Biba77]) must first precisely constrain the domain of control which they will attempt to enforce and then provide carefully calculated distributions of categories with "conventional" interpretations which enable the simple Biba controls to "enforce" the required behaviour.

In contrast [Clark87] attempts to express the problems of the full generality of appropriateness, expressed in their model as Separation of Duty on well-formed transactions. We believe their model should be viewed as a means of expressing the notion and not as a model of execution which attempts to provide the generic controls which actually address the problem.

Having high-lighted the problems with the loose use of the word integrity we will now return from our digression and continue with the development of our security concepts. In this we will continue to use the word integrity in the usual way and will only return to the ramifications of the above distinctions when we arrive at the detailed exposition.

Integrity in this loose sense can be expressed as the desire that at the end of the day the thing is done right. As with the simple secrecy or privacy notion of confidentiality addressing this simple requirement can again be fulfilled with high assurance by a policy of "do it yourself if you want it done right". In the same way therefore it is not suprising that in reality this is not practical and we require a basis of trust for delegating tasks.

Therefore a generic integrity policy requires that things are discretely identifiable and attributed to an individual, lays down a basis for establishing trust in individuals, ie they don't damage, render invalid, etc information given to them, and requires that things are not delegated to multiple individuals other than as permitted by properly established trust.

Establishing Trust In Individuals

Thus for both confidentiality and integrity, once we are beyond the "do it yourself" syndrome, we are therefore concerned with establishing trust in individuals.

To understand all the paraphenalia of Trustworthy Computer bases, security policies, etc and their relationships to this simple concept of trust in individuals we must return to basic simple definitions and concepts.

The requirement for security policies, either military confidentiality based or commercial integrity based, springs from the real world requirement for delegation of tasks to others while wishing to retain responsibility for those tasks.

"I am responsible to the nation for ensuring its security" or "I am responsible to the shareholders for ensuring the profitability of the company" are duties and responsibilities found in the real world of supreme commanders and company directors. In both cases the tasks involved are of such magnitude that obviously whole armies or staffs are employed each member of which having an individual task which are collectively organised to fulfill their leaders declared aim. However, failure of all or part of this collective organisation such that the goal is not achieved results in the demise of the company director or commander and not the court martialling of squaddies or the sacking of the teaboy.

This property of delegation, that responsibility is retained while control is lost, gives rise to the requirement from the delegator that he attains some assurance, guarantee, faith etc that the system instigated by him will not fail due to anothers malicious or negligent actions.

Obviously, this assurance can never be absolute for the following fundamental reason, fallible humans are involved. Ultimately even if the delegator did do the task himself it might still be incorrect. Even if the instigated system contains an element of automation which attempts to constrain human users by enforcement of policy, computers and their programs are built and written by delegated human beings thus moving and not solving the problem.

Given an absence of absolutes what are acceptable relative assurances?

Studying existing techniques leads us to state that the fundamental aspect of all security is the notion of Separation of Duty. Many example of this can be found.

On the grand scale concepts such as accreditation and certification of computer systems stress the importance of the "independence" of the evaluator. On the small scale formal specifications of systems are required so that the "proof of refinement" or mapping functions between orthoganal representations of a system can be carried out according to the "independent" "widely accepted" techniques of mathematics.

We claim that this technique is even the basis of the military models which are often presented as "absolute" definitions of security. We make this claim because in practice such systems are ultimately dependent on the correct assignment of "clearances" and "classifications" to individuals and data. For the former this is carried out by specialised security staffs and is once again separation of duty. For the latter users are commonly "trusted" to correctly assign initial classifications, a total breakdown of separation of duty, or in very high security applications are assumed to generate data at their "clearance" level and is later assigned "appropriate" classifications by reviewers, a clear example of separation of duty.

In all these cases the scheme is the same. The delegator cannot be a sufficient expert in all of the minutiae areas which result from delegating to assure himself that he is safe to delegate. So instead he organises the process of delegation itself in a manner which exploits aspects of human nature such that the delegator indirectly gains confidence in the final system from the inbuilt checks and balances introduced in its development and operation. The aspect which is exploited is the conflicting motivation of individual humans. Thus the delegator issues a contract to develop a system and a contract to evaluate the system. By ensuring that the contracts are carried out by groups with no common interest and encouraging a degree of competition between developer and evaluator the delegator is happy that the system is "safe".

This practice exists within the operation of the final system of subtasks.

If invariant procedural aspects of the system provide no discretion to the human users the developer of the system is effectively in control and thus control has only been delegated as far as the developer and not to the end users. The whole purpose of delegation is to utilise the expertise of others who must therefore be left some discretion in their activities within the system.

However, in order to ensure that users do not use this discretion to undesirable ends the system developer must arrange that the system operates in a manner which enforces the notion of separation of duty. First, users exercise their discretion only by means of choice of one of a number of invariant procedural aspects of the system. Second, important operations of the system are arranged as a sequence of selected invariant procedures. Finally, the system is organised such that the choice at key points within the sequence of procedures is exercised by different users.

However, the underlying assumption which gives us assurance of this approach is not simply that different users are involved but that their motivations are sufficeintly divergent that mistakes and/or fraudulent actions do not occur. The normal military and government practices of "vetting" users is precisely this task of establishing an individuals motivation.

Individuals "Motivation"

For commercial or government sensitive but unclassified arenas a notion of security enforcement based on individual background checks to establish an individuals motivation may be considered unacceptable because of concerns for individual privacy and civil rights issues. Clark-Wilson have shown however that in the commercial sector this same goal is achieved in practice without background checks because of implicit assumptions which can be formed about an individuals "motivation".

The example that springs to mind is that a highly paid, career oriented manager of a Bank is unlikely to collude with lowly paid counter clerks in a petty fraud. Thus if transactions by the clerks require a final counter signature by the manager, who is known to make random spot checks on the veracity of the transactions, fraud by counter clerks is inhibited. Major fraud by the manager alone is prevented by the Bank organisation which does not allow highly placed individuals to carry out the basic transactions, only to countersign and check them. Major fraud by the manager in collusion with his staff is also prohibited by the paradoxical motivation of the clerks not to cooperate, "The big fish never go to prison, its always the little guy who carries the can".

Total Security

This notion of separation of duties amongst individuals who, by background check, assumptions about motivations, or some combination of the two, can be assured of carrying out a task without collusion to violate policy is only applicable to ensuring the integrity of tasks. Confidentiality cannot be ensured by this means. For confidentiality it only takes one subversive or accidental action by one individual to disclose a secret thus the more individuals that acquire knowledge of it the greater the chance of disclosure. Conversely for integrity, individuals able to complete the whole task may be subverted or careless thus by ensuring that many individuals must cooperate in a highly coordinated manner the chance of undesirable results is reduced.

Because of this divergence of properties it may not seem clear how we claim that these two aspects, confidentiality and integrity, together with assurance are all underpinned by the single notion of separation of duty, thus allowing them to be coordinated into a single coherent notion of security.

The answer is that it is not easy to see the relationship, and more importantly accept the true relationship, because of the emphasis which is placed on the artifical distinction between commercial and military security. We say that this is an artifical distinction because there are aspects of any commerical venture which require confidentiality, hence "commercial-in-confidence" and "trade secrets". Similarly, the military also require that they don't lose the war through failure to act and function correctly in spite of foiling the enemies spies. The obvious relationship is that the only reason for wanting a task of information processing to be carried out, military or commercial, presupposes that it will be done correctly, misleadingly ascribed as the only commercial imperative, and that it is in addition to this that we require it to be carried out in confidence, misleadingly ascribed as the only military imperative.

This is difficult for those of a military theoretical background to accept because of the common view that we must have confidentiality at all costs and integrity is an optional add-on. The futility of this view in practice can easily be shown because of inevitable requirements which arise in such military based confidentiality secure system. In such a system if we are to avoid "doing it ourselves" but at the same time retain assurance that confidentiality is upheld, we are forced to implement the controls so as to make the worst case assumption of the behaviour of individuals in terms of the sensitivity labels of information they produce from initially labelled information. For this reason there is a constant need to "trim" overclassified, generated information. Additionally, if the system in its application role within an organisation is taken into account there also arises needs to change the sensitivity labels of information in the machine to reflect the changed perceptions of the environment.

For these reasons there is a requirement that individuals can change the control mechanisms that are implemented to effect controls on individuals. There is obviously a requirement for maintaining the integrity of these controls in the sense that an inappropriate or fraudulent change is not made. In other words in practice integrity control is a prerequisite for implementing real world confidentiality control.

For our notion of total security we have therefore adopted the approach of describing security as the simple real world requirement that a job of work is carried out with respect to privacy considerations, in a correct manner, and that it is done only if it is in some sense "appropriate". These three aspects of security are referred to as Confidentiality, Integrity and Appropriateness.

Our notion of confidentiality in its application to moment to moment access control decisions within the system is unchanged from the typical current approach of a lattice of labels and a dominates relation together with axioms that subject and objects are labelled and information flows subject to the dominates relation.

However, because we are combining at a more fundamental level axioms of integrity and appropriateness, we differ from current approaches in that we do not have further axioms requiring constraints which render the first set of axioms in some sense watertight, absolute controls. Thus concepts such as the changing of subjects and objects classifications are not in violation of our concept of confidentiality requiring "trusted processes" etc but are instead subject to constraints such as "Is it appropriate to change this subject/object to this level" etc.

Our notion of integrity, as alluded to earlier in our digression on commercial policies, is limited to the simple notion of the correctness of a thing in terms of it having a valid state, together with supporting notions of the preservation of such state properties.

Our notion of appropriateness is reserved for the "fitness of an action" aspects of the usual notion of integrity and is expressed in terms of the wider context of the relationship between the internal valid system state and the external real world state as introduced by [Clark87].

Given this intuitive background and the above restrictions in the connotations of terms in their use for the remainder of this document we will now turn to formalising these notions.

3. SECURITY POLICY MODEL

A system is modelled as a set of Entities, which can be thought of as repositories of information. It is our intention to model the behaviour of the system and not the constraints required of an underlying protection mechanism. For this reason we introduce a model element which corresponds to the data that is contained within Entities. This element is the set of Attributes. The Entity/Attribute paradigm can be used for all degrees of granularity, from an integer value stored in a memory location to a file in a directory.

The state of the system can be fully captured by the relationship between these Entities and Attributes.

We establish our notion of security as a property of state transitions concerning the "flow of information". Information is modelled explicitly in the model as the set of Attributes, representing information encoded as data. There is also an implicit encoding of information within the structure of the model concerned with the relationships of Entities and Attributes within the state. In both cases *examination of the state before and after a transition* allows us to deduce the flow of information between Entities.

For example, consider three Entities, A, B and C, and the attributes, 6, 7 and 42, in the following state transition.

$$\{ A \mapsto 6, A \mapsto 7, B \mapsto 6, C \mapsto 7 \} \rightarrow \{ A \mapsto 6, A \mapsto 7, A \mapsto 42, B \mapsto 6, C \mapsto 7 \}$$

There are a number of "logical explanantions" of this event in terms of information flow. "A" may have carried out a multiplication of its own attributes in which case there is no information flow between entities. Or A may have observed B and Cs attributes in a multiplication in which case information flows from B and C to A. Or A may have observed B or Cs attribute and one of its own in which case information flows from B or C to A. Or B may have carried out the multiplication reading Cs attribute, its own, and placing the result directly in A in which case information flows from B and C to A, etc,etc.

To distinguish which of these flows actually occured in the transition we require more structure in the model.

We can be sure that A is the recipient of the explicit information flow represented by the data attribute "42" by comparing the before and after states. However, this does not allow us to precisely identify those entities which were the source of this flow. The transition request must therefore explicitly identify these entities. We give this set of entities the suggestive name of the "observed" entities. Recognising that if something is observed something must be doing the observing we can also suggestively name the set of entities which are the destination of information flows the "observers".

As we are modelling a purposeful machine rather than random events of nature we must obviously identify one or more entities as the instigators of state transitions. Thus we model state transition "requests" which identify some entities as the "requestors" of the state transition in addition to identifying the "observed" entities.

Given a request, identifying "requestors" and "observed", and the before and after states of a transition, allowing "observers" to be determined, we have sufficient structure in the model to distinguish all possible scenario's of the possible flows in the example and of a system in general.

Before proceeding further we will first introduce the formal machinery necessary to discuss the details of our policy approach unambiguously using the Z notation [Spivey87]. First the set of entities and attributes are parachuted in as basic types.

$$[E, A]$$

We identify various sub-sets and members of the set of attributes for use in expressing our notions of security. The meaning and use of these will become apparent later.

$$CLASS, TRUST, ROLE : \mathbb{P}\ A$$

$$dont\_signal, faithful, creator : TRUST$$

The state of the machine, V, is represented by a schema. This comprises the relation between entities and attributes which is structured into a number of named relations and functions, the use of which will again become clear later.

V

$$\begin{aligned} Class &: E \pfun CLASS \\ Trust &: E \rightarrow TRUST \\ Role &: E \pfun ROLE \\ Conflict &: E \rightarrow ROLE \\ Other &: E \rightarrow A \end{aligned}$$

Requests, R, are modelled as a set of entities which are "observed" together with the set of "requestors" responsible for the request.

R

$$observed, requestors : \mathbb{P}\ E$$

The results of a request is a decision together with a possible change of state.

$$D \triangleq \{ \text{"yes"}, \text{"no"} \}$$

We introduce some useful operators as language extensions.

[X]

$$\begin{aligned} &\_\supset\_ : \mathbb{P}\ X \rightarrow \mathbb{P}\ X \\ &\_\supseteq\_ : \mathbb{P}\ X \rightarrow \mathbb{P}\ X \\ &(\_\uparrow\_) : (\mathbb{P}\ X \times \mathbb{P}\ X) \rightarrow \mathbb{P}\ X \end{aligned}$$

$$\begin{aligned} &\forall\ x, y : \mathbb{P}\ X \bullet \\ &\quad x \supset y \Leftrightarrow y \subset x \\ &\quad x \supseteq y \Leftrightarrow y \subseteq x \\ &\quad \forall\ x, y : \mathbb{P}\ X \bullet x \uparrow y = x \cup y \setminus x \cap y \end{aligned}$$

This defines the righthand versions of the standard subset and strict subset relations for the benefit of our type checker which only defines the lefthand versions in its basic library. The $\uparrow$ function is our definition of the standard symmetric set difference function which is absent from our basic library.

In defining our system we will require a number of relations and functions which enable us to characterise state transitions in terms of the differences between states. These language extensions are introduced in one fell-swoop and are essentially quite trivial. If one thinks of the state as a single relation, say F, between entities and attributes instead of the structured set of relations actually given then the meaning of these state functions and relations is what one would intuitively say about the domain and range of F with the standard functions. The state relations use the same symbol as the standard relation except they are in **bold**. If our type checker supported overloading of symbols we would in fact simply do so.

First the state relations,

$$\mathbf{dom}\_ : V \to \mathbb{P}\, E$$
$$\mathbf{rng}\_ : V \to \mathbb{P}\, A$$
$$\_\sqsupseteq\_,\ \_\mathbf{=}\_,\ \_\neq\_,\ \_\sqsupset\_,\ \_\sqsubset\_,\ \_\sqsubseteq\_ \ : V \leftrightarrow V$$

$$\forall u,v : V \bullet$$
$$\mathbf{dom}\ v = \mathbf{U}\ \{dom\ v.Class,\ dom\ v.Trust,\ dom\ v.Role,\ dom\ v.Conflict,\ dom\ v.Other\ \}$$
$$\mathbf{rng}\ v = \mathbf{U}\ \{rng\ v.Class,\ rng\ v.Trust,\ rng\ v.Role,\ rng\ v.Conflict,\ rng\ v.Other\ \}$$
$$u \sqsupseteq v \Leftrightarrow (u.Class \sqsupseteq v.Class \wedge u.Trust \sqsupseteq v.Trust \wedge u.Role \sqsupseteq v.Role \wedge u.Conflict \sqsupseteq v.Conflict \wedge u.Other \sqsupseteq v.Other)$$
$$u = v \Leftrightarrow u \sqsupseteq v \wedge v \sqsupseteq u$$
$$u \neq v \Leftrightarrow \neg(u = v)$$
$$u \sqsupset v \Leftrightarrow u \sqsupseteq v \wedge u \neq v$$
$$u \sqsubset v \Leftrightarrow v \sqsupset u$$
$$u \sqsubseteq v \Leftrightarrow v \sqsupseteq u$$

and then the state functions.

$$(\_\cap\_),\ (\_-\_),\ (\_\uparrow\_) : (V \times V) \to V$$
$$(\_\lhd\_) : (\mathbb{P}\ E \times V) \to V$$

$$\begin{aligned}
&\forall\ u,v,w : V;\ e : \mathbb{P}\ E \bullet \\
&\quad u \cap v = w \Leftrightarrow\ w.Class = u.Class \cap v.Class \wedge \\
&\qquad\qquad w.Trust = u.Trust \cap v.Trust \wedge \\
&\qquad\qquad w.Role = u.Role \cap v.Role \wedge \\
&\qquad\qquad w.Conflict = u.Conflict \cap v.Conflict \wedge \\
&\qquad\qquad w.Other = u.Other \cap v.Other \\
&\quad u - v = w \Leftrightarrow\ w.Class = u.Class \setminus v.Class \wedge \\
&\qquad\qquad w.Trust = u.Trust \setminus v.Trust \wedge \\
&\qquad\qquad w.Role = u.Role \setminus v.Role \wedge \\
&\qquad\qquad w.Conflict = u.Conflict \setminus v.Conflict \wedge \\
&\qquad\qquad w.Other = u.Other \setminus v.Other \\
&\quad u \uparrow v = w \Leftrightarrow\ w.Class = u.Class \uparrow v.Class \wedge \\
&\qquad\qquad w.Trust = u.Trust \uparrow v.Trust \wedge \\
&\qquad\qquad w.Role = u.Role \uparrow v.Role \wedge \\
&\qquad\qquad w.Conflict = u.Conflict \uparrow v.Conflict \wedge \\
&\qquad\qquad w.Other = u.Other \uparrow v.Other \\
&\quad e \lhd v = w \Leftrightarrow\ w.Class = e \lhd v.Class \wedge \\
&\qquad\qquad w.Trust = e \lhd v.Trust \wedge \\
&\qquad\qquad w.Role = e \lhd v.Role \wedge \\
&\qquad\qquad w.Conflict = e \lhd v.Conflict \wedge \\
&\qquad\qquad w.Other = e \lhd v.Other
\end{aligned}$$

State transitions are described by a schema, W, which capture the state machine concepts of a request in a state resulting in a response and a new state.

W

$$\begin{aligned}
&r? : R \\
&d! : D \\
&v,\ v' : V \\
&observers,\ modified,\ modified_controls,\ changed_controls : \mathbb{P}\ E \\
&\hline
&\emptyset \subset r?.requestors \subseteq r?.observed \subseteq \mathbf{dom}\ v \\
&changed_controls = \mathbf{dom}(\ v.Class \uparrow v'.Class\ ) \\
&\qquad\qquad \cup\ \mathbf{dom}(\ v.Conflict \uparrow v'.Conflict\ ) \\
&\qquad\qquad \cup\ \mathbf{dom}(\ v.Role \uparrow v'.Role\ ) \\
&\qquad\qquad \cup\ \mathbf{dom}(\ v.Trust \uparrow v'.Trust\ ) \\
&observers = \mathbf{dom}(\ v \uparrow v'\ ) \cup r?.requestors \\
&modified_controls = changed_controls \cap \mathbf{dom}\ v \cap \mathbf{dom}\ v' \\
&modified = observers \cap \mathbf{dom}\ v \cap \mathbf{dom}\ v'
\end{aligned}$$

This schema defines a number of sets of entities which are used in the policy axioms to capture our various notions of security. It is therefore very important that one clearly understands the characteristics of each of these sets.

r?.requestors and r?.observed

These sets are identified in a request, R, schema. However, to usefully model reality these sets cannot simply be arbitrary and must conform to the following minimal constraints.

$\emptyset \subset$ r?.requestors $\subseteq$ **dom** v
Clearly, transitions can only occur if they are requested hence r?.requestors must not be an empty set. Equally, a requestor can only request a transition if it exists.

r?.observed $\subseteq$ **dom** v
Similarly, an entity can only be observed in a transition if it exists in the current state.

r?.requestors $\subseteq$ r?.observed
Finally, because a requestor can influence the final state by choosing whether or not to invoke a transition, there is an implicit flow of information from requestors to observers as well as the explicit flow from the observed entities. We account for this is the model by insisting that the requestors are a subset of the observed entities.

observers = **dom** v $\triangleleft$ (v $\oplus$ v') $\cup$ r?.requestors

This set can be defined in terms of the differences between the before and after states of the transition. The symmetric set difference operator on states captures all notions of change in a state including deleting and creating entities as well as the more mundane non-destructive modification of the mapping between existing entities and attributes. Because our model includes the notion of multiple requestors, for n-man rules, etc, there are possible information flows between requestors. For example, if an entity can carry out a transition on its own but not in conjunction with some other requestor it can infer something about the security controls of that other requestor. In order to account for this we insist that observers includes the requestors in its definition.

modified = observers $\cap$ **dom** v $\cap$ **dom** v'
modified is simply observers with entities which were created or destroyed in the transition excluded. We tend to make this distinction because of the fundamental differences in the security properties of creation and deletion versus modification of entities.

changed_controls
This set is analogous to observers except that it is only considering changes to the "control aspects" of the state and does not concern itself with changes to "ordinary data" parts. Thus, like observers, changed controls encompasses creation and deletion of entities.

modified_controls = changed_controls $\cap$ **dom** v $\cap$ **dom** v'
Again this is analogous to modified restircted to the control aspects of the state. Thus, like modified, it is not concerned with creation and deletion.

An appearance of the state machine is defined by the sequence of states arising from the application of a sequence of transition requests, given an initial state and a set of rules. A system is defined by the set of appearances from a particular initial state and set of rules.

Appearance

$z_0 : V$
states : seq V
inputs : seq R
output : seq D
rules : ℙ W

---

states 1 = z_0
states = # inputs + 1
inputs = # output
∀ n : 1 .. #inputs •
output n = "yes"
⇒
∃ r : rules •
r.v = states n
r.v' = states (n+1)
r.r? = inputs n
output n = "no"
⇒
states n = states (n+1)

We believe that the model of execution presented above is sufficiently powerful to describe any computer system. Thus we will use this abstraction of a computer to address the question of what does it mean for a computer system to be secure, without considering particular implementation mechanisms.

Confidentiality

The information that war starts tomorrow is secret. However information is ephemeral and uncontrollable and must always be encoded in some medium, such as soundwaves in air, characters on paper etc. An encoding such as the string "War Starts Tomorrow" is not a secret, it is just a sequence of characters. Hence, if we require a system which limits the possesion and distribution of information the best approximation we can achieve is to control the distribution and possesion of encodings of information. These controls work by denying access on the basis of labelled containers of such encodings where the label reflects their information content.

In the model containers are represented by entities and encodings such as strings by attributes. However, while information should be encoded directly as the value of an attribute, it may also be encoded indirectly in the distribution of attributes amongst entities or in the existence of entities.

The first case is the obvious encoding of information as data, which is something that can be controlled in terms of where it is stored. Therefore we can protect such information in the usual way by labelling containers.

In the second case information is encoded in a less obvious way using the distribution of attributes rather than their values. However this is still in terms of where attributes are stored and so can be controlled by "worst-case" labelling of containers. However, labels are themselves attributes and because of their interaction with the state transition rules information can be encoded in them. In general, whenever a protection regime is introduced it becomes possible to encode information in its mechanisms. Control of such "signalling channels" will be discussed shortly.

The third case is where information is encoded in the existence of entities. This signalling channel is another example of the general rule and arises because attributes may only be accessed via entities, which is a weak protection regime. This channel cannot of course be controlled by labelling because it is not concerned with storing attributes in entities.

There are two possible solutions to the problems of signalling channels which arise through encoding information in the protection regime mechanisms. One solution is to prohibit "interference" caused by changes in such mechanisms. The alternative is to allow this functionality and ignore the channels. Neither of these are acceptable as system wide solutions. Prohibition cannot be used because such encoding arises not only through illegal attempts to bypass the security controls but also as side effects of legitimate operations. Ignoring such channels is not viable in high security applications where such attacks form a real threat.

Applying the two solutions selectively, according to some suitable criterion, is therefore desirable. We choose to model this criterion as degrees of trust in the requestors. That is trusted requestors are allowed to create and delete entities or change entities' classifications because they do not exploit the unavoidable signalling channels which arise as a side effect. However it is assumed that untrusted requestors will attempt to exploit such side effects and so are prevented from making such changes.

In summary, confidentiality is about using labels to control the flow of information encoded in the contents of containers, and using trust to control the flow of information encoded in the mechanisms of the protection regimes.

In order to model the basic concept of labelling containers we need to create the usual paraphenalia of the lattice of classification labels. In the formal model this role is served by the set of attributes CLASS and the dominates relation, $\geq$ (or $\leq$ where convenient), together with typical supporting definitions of useful least upper bound operators etc.

$$\begin{aligned} GLB, LUB &: \mathbb{P}\ CLASS \rightarrow CLASS \\ \_\geq\_ &: CLASS \leftrightarrow CLASS \\ \_\leq\_ &: CLASS \leftrightarrow CLASS \end{aligned}$$

Labelling of entities with these classifications is achieved by the partial function "Class" element of the state.

In order to model the concept of selectively allowing access to the signalling channels we need a notion of trust. In general an entity may be trusted differently with respect to various aspects of the system. Therefore in the formal model we define an attribute for each of these aspects, collectively called TRUST, and a relation, "Trust", which ascribes appropriate trust attributes to entities.

For confidentiality the only aspect affected by trust is whether an entity exploits a signalling channel. Entities which do not exploit the channel have the dont_signal attribute.

The confidentiality control expressed in terms of the overt flow of information between labelled containers is simply that in a transition the classification of all modified entities dominates the classification of all observed entities.

Confidentiality1

$$W$$

$$GLB\ v.Class(\ modified\ ) \geq LUB\ v.Class(\ r?.observed\ )$$

This is analogous to the Bell-LaPadula *-property except that it is expressed in terms of overt information flow in a state transition.

Some nuances of this approach are worthy of comment.

The Bell-LaPadula *-property was introduced in their modelling technique to ensure the watertightness of their notion of security. This interpretation must not be carried across to our model under any circumstances. This axiom is concerned soley with the overt transfer of information amongst labelled containers. It does not say anything about the transfer of information between un-labelled conatiners or a mixture of labelled and un-labelled containers.

This axiom is also not concerned with the covert transfer of information due to the creation and deletion of entites or changes in their classification.

All of these properties may appear to be "problems" in that they are counter to *ones common notion of security*. However it should be noted that they are all about the appropriateness of a transition. Therefore these aspects are dealt with by that aspect of security, rather than by creating a raft of further supporting confidentiality axioms.

The second component of confidentiality, that signalling paths are not used by untrusted software, is expressed as follows.

Confidentiality2

$$W$$

$$\begin{array}{l} (\mathbf{dom}\ v' \neq \mathbf{dom}\ v \vee \\ \quad v.Class \upharpoonright v'.Class \neq \emptyset \vee \\ \quad v.Conflict \upharpoonright v'.Conflict \neq \emptyset) \\ \Rightarrow \\ \quad \forall\ r : r?.requestors \bullet dont\_signal \in v.Trust(\{r\}) \end{array}$$

The first disjunct captures the signalling channel via entity creation and deletion while the second and third capture the channels through the changing of an entities "observe" and "modify" control attributes.

An interesting point to notice about this axiom is that it doesn't prevent requestors from modifying their own, or other entities, trust status. As with the first axiom this apparent problem is concerned with the appropriateness of such a modification.

Finally our overall definition of confidentiality is expressed thus:

$$Confidentiality \triangleq Confidentiality1 \wedge Confidentiality2$$

Integrity

We define the concept of integrity as a property of state; that it is "correct". Obviously in a state machine we require all states to be correct in order that the machine is correct overall.

In this model we have only one simple requirement for correctness, which is that the confidentiality controls can at all times be correctly applied. This requires the following integrity constraint on all transitions.

Integrity

$$W$$

$$(observers \cup r?.observed) \cap \mathbf{dom}\ v \subseteq dom\ v.Class$$
$$(observers \cup r?.observed) \cap \mathbf{dom}\ v' \subseteq dom\ v'.Class$$

Effectively, all transitions rules are type operators which only act on classified entities and deliver classified entities.

Appropriateness

Appropriateness, defined in Clark-Wilson terms as that aspect of "data integrity" related to the correspondence between the systems internal view of the world and the true external state, is a property of a system which we cannot enforce within the system. However, we can construct the system such that its operation by human users, undeniably in the "real" world, will tend to favour correspondence as opposed to deviation from the real world.

The feature of a system so constructed is its use of a technique, called by Clark-Wilson "Separation of Duty", which exploits the conflicting motivation of the human users. In order to capture the concept of confliciting motives users are assigned to roles. Many users can be assigned to a role but a single user can only have one role. This is the purpose of the attribute set ROLE and the function "Role" in the model.

Separation of Duty is then achieved by forcing users with confliciting roles to participate in n-man operations.

In our modelling approach a request originates from a set of requestors. Active entities in the model represent software invoked on behalf of the human user and endowed with the user's "authority" to carry out some task. We recognise that not all software can be trusted not to abuse these "rights" and carry out additional or alternative tasks which the user did not intend when invoking the software. This is the Trojan Horse problem. In the model we ascribe the trust attribute, "faithful", to those entities which are always faithful proxies, ie Trojan Horse free. In the face of possible Trojan Horses, enforcing the n-man rule cannot simply be enforced by requiring n-requestors, but instead requires us to count only the "faithful" proxies.

To capture the conflict aspects of the n-man rule we need to restrict state transitions to faithful requestors acting on behalf of individuals about whose motivation some assumption of conflict can legitimately be made.

Human intuition, which we may never be able to logically model, tells us that a clerk or a bank manager on their own may be tempted to defraud the bank. The n-man rule means that two clerks or two managers are less likely to collude to defraud the bank. However, for social reasons, we are inclined to believe that a clerk and a manager, while still only fulfilling a two-man rule are far more unlikely to collude. This role based control can be implemented with access control lists. To be able to capture this degree of control in the access control lists we need to be able to associate appropriately conflicting roles with entities. This is the purpose of the relation "Conflict".

Our axiom for maintaining appropriateness is thus the Separation of Duty axiom.

Separation_of_Duty

W

$$\forall e : \text{modified\_controls} \bullet v.\text{Role}(\text{faithful\_requestors}) \supseteq v.\text{Conflict}(\{e\})$$

where

$$\text{faithful\_requestors} \triangleq \{ r : r?.\text{requestors} \mid (r \mapsto \text{faithful}) \in v.\text{Trust} \}$$

This axiom applies only under two assumptions. Firstly, that only the modification of existing entities is under consideration and, secondly, that the initial assignment of Conflict roles to entities correctly captures truly conflicting roles. Clearly, this is a bootstrapping problem which requires alternative criteria for trusting the initial states "appropriateness" and, if the creation of entities is to be modelled, trusting subsequent requestors to correctly set up "appropriate" controls on newly created entities.

In our model, as in all modelling approaches, the appropriateness of the initial state cannot be defined, however, we can define a notion of trust to ensure the appropriateness of the controls on newly created entities.

In the particular model in this paper the axiom which we formulate is that at least one requestor is appropriately trusted to change the controls with respect to new entities.

Trusted_Creation

W

$$\text{dom } v' \setminus \text{dom } v \neq \emptyset$$

$$\Rightarrow$$

$$\text{creator} \in v.\text{Trust}(r?.\text{requestors})$$

Thus overall we require the following axiom to ensure the appropriateness of changes made to the system.

$$\text{Appropriateness} \triangleq \text{Separation\_of\_Duty} \wedge \text{Trusted\_Creation}$$

Having introduced our model of execution and the notions of security which we can express about its behaviour we now turn to a description of the state transition rules.

The Rules

It is our intention to describe rules in terms of a small number of broad classes of state transitions rather than in terms of a large number of individual transitions. We can define a class in terms of the common security constraints shared by all transitions in that class.

State transitions are characterised by their effects on the state relations of the before and after states. The essential model of execution is that a group of entities, the observers and observed, are involved in a transition and that the state relations in respect of all other uninvolved entities are unaffected by the state transition.

The most obvious effect of a transition on a state is in terms of the gain and/or loss of elements in the state relations. The elements lost and/or gained in a relation can be further categorised as to whether they involve new and/or rearranged relationships of attributes with existing entities, or whether they involve the loss and/or gain of entities.

These three classes of transitions, gain of entities, loss of entities and change of attributes, are the only discriminants of the security contraints which must be applied to a transition and do not impose any discernible restriction on the variety of state transitions which can be used to model real events within these classes.

Thus these three rules are sufficient to model a real system.

The Change Attribute Rule

Change of attributes is defined as the class of transitions where the state changes (2) in a manner which does not involve the loss and/or gain of entities (3).

ChangeAttribute

W

d! = "yes" (1)

v' ≠ v (2)

dom v' = **dom** v (3)

(observers ∪ r?.observed) ∩ **dom** v ⊆ dom v.Class

(observers ∪ r?.observed) ∩ **dom** v' ⊆ dom v'.Class (4)

GLB v.Class(modified) ≥ LUB v.Class(r?.observed) (5)

(v.Class ↑ v'.Class ≠ ∅) ∨
(v.Conflict ↑ v'.Conflict ≠ ∅) (6)

⇒
(∀ r : r?.requestors • dont_signal ∈ v.Trust({r}))

∀ e : modified_controls •
v.Role(faithful_requestors) ⊇ v.Conflict({e}) (7)

where
faithful_requestors ≙ { r : r?.requestors |
(r↦faithful) ∈ v.Trust } (8)

This rule upholds Confidentiality1 by virtue of the predicate of the axiom being present in the rule (5).

The preconditions of the rule prohibit creation of entities so the first disjunct of Confidentiality2 is always false for this rule. However, as modification of controls is not barred by the preconditions of this rule the second and third disjuncts of Confidentiality2 can be satisfied. Thus Confidentiality2 can be upheld by including the modified predicate of the axiom in the rule (6).

Integrity is upheld by virtue of the predicate of the axiom being present in the rule (4).

Separation of Duty is satisfied by virtue of the predicate of the axiom being present in the rule (7 & 8).

For the Trusted_Creation axiom the precondition of the predicate is always false because the precondition of this rule prohibits the creation of any new entities. Thus this rule trivially upholds the implication of that axiom.

Thus this rule upholds Confidentiality, Integrity and Appropriateness.

The Entity Gain Rule

Gain of entities is defined as the class of transitions where the changes to the state relations involve a gain only of elements (2) and these new elements involve no new attributes (4) and only new entities (3).

EntityGain

W

d! = "yes"	(1)
v' ⊐ v	(2)
dom (v' − v) ∩ **dom** v = ∅	(3)
rng v' = **rng** v	(4)
(observers ∪ r?.observed) ∩ **dom** v ⊆ dom v.Class (observers ∪ r?.observed) ∩ **dom** v' ⊆ dom v'.Class	(5)
GLB v.Class(modified) ≥ LUB v.Class(r?.observed)	(6)
∀ r : r?.requestors • dont_signal ∈ v.Trust({r})	(7)
dom v'\ **dom** v ≠ ∅ ⟹ creator ∈ v.Trust(r?.requestors)	(8)

This rule upholds Confidentiality1 by virtue of the predicate of the axiom being present in the rule (6).

As the preconditions of the rule always imply the first disjunct of Confidentiality2 the condition of the predicate is included in the rule (7) thereby satisfying Confidentiality2.

Integrity is upheld by virtue of the presence of the predicate of the axiom in the rule (5).

The preconditions of this rule mean that no entities are modified therefore Separation of Duty is trivially true.

The preconditions of the rule imply that Trusted_Creation may apply. As the rule includes the predicates of the axiom it trivially supports this axiom.

Thus this rule upholds Confidentiality, Integrity and Appropriateness.

The Entity Loss Rule

Loss of entities is defined as the class of transitions where the changes to the state relations involve a loss only of elements (2) and no loss or gain of attributes (4) and the entities involved in these lost elements are absent in the new state (3).

EntityLoss

W

$$d! = \text{"yes"} \quad (1)$$

$$v' \sqsubset v \quad (2)$$

$$\mathbf{dom}(v - v') \cap \mathbf{dom}\ v' = \emptyset \quad (3)$$

$$\mathbf{rng}\ v' = \mathbf{rng}\ v \quad (4)$$

$$(observers \cup r?.observed) \cap \mathbf{dom}\ v \subseteq dom\ v.Class$$
$$(observers \cup r?.observed) \cap \mathbf{dom}\ v' \subseteq dom\ v'.Class \quad (5)$$

$$GLB\ v.Class(\!|\ modified\ |\!) \geq LUB\ v.Class(\!|\ r?.observed\ |\!) \quad (6)$$

$$\forall\ r : r?.requestors \bullet dont\_signal \in v.Trust(\!|\{r\}|\!) \quad (7)$$

$$\forall\ e : modified\_controls \bullet$$
$$v.Role(\!|\ faithful\_requestors\ |\!) \supseteq v.Conflict(\!|\ \{e\}\ |\!) \quad (8)$$

where

$$faithful\_requestors \triangleq \{\ r : r?.requestors \mid (r \mapsto faithful) \in v.Trust\ \} \quad (9)$$

This rule upholds Confidentiality1 by virtue of the predicates of the axiom being present in the rule (6).

As the preconditions of the rule always imply the first disjunct of Confidentiality2 the condition of the predicate is included in the rule (7) thereby satisfying Confidentiality2.

Integrity is upheld by virtue of the presence of the predicate of the axiom in the rule (5).

Separation of Duty is satisfied by virtue of the predicate of the axiom being present in the rule (8 & 9).

The precondition of the Trusted_Creation axiom isfalse because the precondition of the rule prohibits the creation of any new entities. Thus this rule trivially upholds that axiom.

Thus this rule upholds Confidentiality, Integrity and Appropriateness.

Overall Security

A secure system is defined as those appearances whose rules uphold the axioms of confidentiality and appropriateness. We have defined such a set of rules above which can be summarised thus:

$$Rules \triangleq ChangeAttribute \vee EntityGain \vee EntityLoss$$

Thus a secure system is defined as follows:

Secure_Appearances

Appearance

$\forall$ r : rules •
r ∈ Rules
r ∈ *Confidentiality*
r ∈ Integrity
r ∈ Appropriateness

BASIC SECURITY THEOREM AND PROOF

The state machine, rules and axiom set given define a secure system where security is defined in terms of confidentiality, integrity and appropriateness. The model serves to both define and exhibit the properties of integrity and appropriateness so there is little we can do to assure the correctness of these concepts other than appeal to intuition and the simplicity of the model.

However, for confidentiality there are other definitions and demonstrations of the concept. Therefore we would like to offer the following analysis of our model, in terms of one of these other definitions, in order to provide an orthoganal "proof" that our axiom set for confidentiality is necessary and sufficient.

The analysis which we will perform is a non-interference analysis.

The standard MLS policy is usually expressed in this formalism as the following set of *non-interference* assertions:

$$\{ u :| v | Level(u) \geq Level(v) \}$$

where u and v are members of the set of users and Level is a function which gives the fixed security level of a user.

In the non-interference approach this notion is formalised by defining a state machine with users, states, commands and next and out functions etc.

There are some problems with the standard notion of non-interference in respect of our model. We work with groups of users executing commands instead of single users thus allowing us to consider the notion of a users security level being changed by consensus.

The original paper extolling this technique [Gougen82] claimed to work with "dynamic polices", ie where the security level of a user was not fixed. Private communication with Rushby indicates that all was not well with this paper and that a lot of the ideas in it were wrong. Certainly in all non-interference work seen since only fixed level functions users have been used. These subsequent papers, [Gougen84], [Rushby85], [Haigh86], are significant in that they establish the form and use of the "unwinding theorem", which is the only useful form in which formal analysis can actually be performed.

An interference analysis of our model should show that the rules of our model uphold non-interference between entities as in any other MLS policy except where explicitly excepted by the dont_signal trust mechanism.

I will repeat the basis of [Rushby85] in terms of our model as follows.

A machine M is composed of

- a set V of states with an initial state $v^0 \in V$,
- a set E of entities
- a set C of transition rules

together with the state transiton function next and output function out:

- $next : V \times (\mathbb{P}\, E \times C) \rightarrow V$
- $out : V \times (\mathbb{P}\, E \times C) \rightarrow D$

where $\mathbb{P}\, E \times C$ corresponds to our normal request, R, notation. Throughout this section we will use the R notation for convenience and to more easily relate to the existing model.

- $next : V \times R \rightarrow V$
- $out : V \times R \rightarrow D$

The idea here is that next(v,r) denotes the next state of the system when the request r is invoked in state s, while out(s,r) denotes the result returned by the request r in state s. The result consists of the "yes"/"no" response of the machine. The next and out functions are implicit in the Z notation when using the "before" and "after" primes, bangs and query decorations.

We derive a function $next^*$:

$$next^* : V \times R^* \rightarrow V$$

(the natural extension of next to sequences of actions) by the equations

$$next^*(v,\Lambda) = v, \text{ and}$$
$$next^*(v,\alpha \circ r) = next(next^*(v,\alpha),r),$$

where Λ denotes the empty sequence and $\circ$ denotes concatenation.

We define the functions do and result:

$$do: R^* \rightarrow V$$
$$result: R^* \times R \rightarrow D$$

by the equations

$$do(\alpha) = next^*(v^0,\alpha),$$

$$result(\alpha,r) = out(do(\alpha),r).$$

The intuition here is that the machine starts off in the initial state v^0 and is presented with a sequence $\alpha \in R^*$ of actions. This causes the machine to progress through a series of states, eventually reaching the state $do(\alpha)$. At that point, the request r is performed and receives the result $result(\alpha,r)$. Observe that this result is merely the final output seen when the system performs r in state $do(\alpha)$; it is not the sequence of intermediate results seen during execution of the action sequence α.

A state $v \in V$ is said to be reachable if there exists an action sequence $\alpha \in R^*$ such that $v = do(\alpha)$.

In order for a certain request r to be said to cause information to flow to subsequent requests, it must surely be the case that results subsequently seen are different from those that would have been seen if the request concerned had not been present. That is, the request r may be said to convey information or interfere with q, if

$$result(\alpha \circ r \circ \beta, q) \neq result(\alpha \circ \beta, q).$$

This is the basis of the non-interference identification of security as the requirement that information may not flow from certain (classes of) requestors to others. We say that there is no information flow from one requestor to another, or that the first is non-interfering with the second, if the results seen by the second requestor are completely unaffected by the presence or absence of operations involving the first. We now state this notion formally:

Definition 1: For $u, v \in E$ and α an action sequence in R^*, define α/u to be the subsequence of α formed by deleting all requests involving u, that is:

$$\Lambda/u = \Lambda,$$

$$(\alpha \circ r)/u = \begin{cases} \alpha/u & \text{if } u \in r.requestors, \\ \alpha/u \circ r & \text{otherwise.} \end{cases}$$

A requestor u is **noninterfering** with requestor v if

$$result(\alpha, q) = result(\alpha/u, q),\ \forall\, \alpha \in R^* \text{ and } q \in R \mid q.requestor = v.$$

A **security policy** is a relation $\not\rightsquigarrow$ on the set of entities E with the interpretation that $u \not\rightsquigarrow v$ indicates that requestor u is required to be non-interfering with v.

A system M is **secure** with respect to a policy $\not\rightsquigarrow$ if u is noninterfering with v whenever $u \not\rightsquigarrow v$. We will generally say simply M is secure in contexts where the relation $\not\rightsquigarrow$ is understood.

We use $\rightsquigarrow$ to denote the complement of $\not\rightsquigarrow$. That is

$$\rightsquigarrow \equiv E \times E \setminus \not\rightsquigarrow$$

where \ denotes set difference. We use $\rightsquigarrow$ rather than $\not\rightsquigarrow$ whenever it is more convenient so to do.

A policy $\not\rightsquigarrow$ is said to be **closed** if its complement $\rightsquigarrow$ is transitive. □

For technical reasons closed policies are required. [Rushby85] now proceeds to show that this is adequate for MLS policies and indeed that all closed policies are MLS. His definition of MLS is that of Feiertag and is defined in the next definition. We depart at this point with a definition of MLS based on state dependent levels.

Definition 2: Let L be a set of security labels with a partial ordering $\leq$ and let **class** : $E \rightarrow L$ be a state dependent function which assigns a **variable** security label to each entity. (The interpretation of class(u) $\leq$ class(v) is that information is permitted to flow from u to v.) Then the **multilevel security (MLS) policy** is:

$$u \not\rightsquigarrow v \text{ iff } \neg(v^0.class(u) \leq v^0.class(v))$$

That is, u must be *non-interfering* with v from state v^0 if information is not allowed to flow from u to v in the initial state. Equivalently,

$$u \rightsquigarrow v \text{ iff } v^0.\text{class}(u) \leq v^0.\text{class}(v) \qquad (1)$$

That is u may interfere with v from state v^0 if information is permitted to flow from u to v in the initial state.

An arbitrary policy given by the relation $\not\rightsquigarrow$ on E is said to be an **MLS-type policy** if a label set L with partial ordering $\leq$ and a function class: $E \rightarrow L$ can be found such that (1) holds. □

Clearly we have:

The above variation of the definition does not effect the remaining development and at this point we return to the [Rushby85] development.

Theorem 3: All MLS-type policies are closed.

Proof: Obvious. □

The converse is also true.

Theorem 4: All closed policies are MLS-type policies.

Proof: Let $\not\rightsquigarrow$ be a closed security policy and $\rightsquigarrow$ its complement. Define a further relation $\leftrightarrow$ on E by

$$u \leftrightarrow v \equiv u \rightsquigarrow v \text{ and } v \rightsquigarrow u.$$

Recall that $\rightsquigarrow$ is reflexive and that, by definition $\rightsquigarrow$ is transitive if $\not\rightsquigarrow$ is closed. It is easy to verify that $\leftrightarrow$ is an equivalence relation.

We identify a label set L with the equivalence classes of $\leftrightarrow$ and use [u] to denote the equivalence class of user u under $\leftrightarrow$. We define a relation $\leq$ on L as follows:

$$[u] \leq [v] \text{ if } \exists \text{ users } x \in [u] \text{ and } y \in [v] \text{ such that } x \rightsquigarrow y.$$

It is easy to see that $\leq$ is a partial order on L (*ie it is reflexive, transitive* and antisymmetric). Finally, we define the function class: $E \rightarrow L$ by

$$\text{class}(u) = [u]$$

It is then easy to verify that

$$u \not\rightsquigarrow v \text{ iff } \text{class}(u) \leq \text{class}(v).$$

It follows that $\not\rightsquigarrow$ is an MLS-type policy. □

In order to verify the security of a system, we suppose that for each state of the system and for each user it is possible to abstract a "view" of the state "as seen by" the user concerned. The verification technique will be to prove that each user's view of the system is unaffected by the actions of users who are required to be non-interfering with him. "Views" cannot be constructed arbitrarily, however, a notion of "internal consistency" is required.

Definition 5: For each entity $u \in E$, let V_u be a set of "private states", and let $abstract_u$ be a function

$$abstract_u : V \rightarrow V_u .$$

For notational convenience, we denote $abstract_u(v)$ by $v[u]$.

A system is **internally consistent** if, $\forall$ reachable states $s,t \in V$ and $r \in R$,

$$s[r.requestor] = t[r.requestor] \Rightarrow out(s,r) = out(t,r).$$

□

Thus if two states s and t "look the same" to a user, he will always obtain identical results when the same operation is performed to each state, provided M is internally consistent.

The definition of security requires that outputs seen by one user are unaffected by operations performed by those other users who are required to be noninterfereing with him. The next result shows that, for an internally consistent system, security is achieved if the view of a user is unaffected by operations performed by other noninterfering users.

Lemma 6: Let $\not\rightsquigarrow$ be a policy and M an internally consistent machine such that, $\forall$ $u,v \in E$, and $\alpha \in R^*$,

$$u \not\rightsquigarrow v \Rightarrow do(\alpha)[v] = do(\alpha/u)[v].$$

Then M is secure with respect to $\not\rightsquigarrow$.

Proof: Since M is internall consistent,

$$\forall\, r \in R \mid r.requestor = v \;\bullet\; do(\alpha)[v] = do(\alpha/u)[v] \Rightarrow out(do(\alpha),r) = out(do(\alpha/u),r)$$

By definition, $out(do(\alpha),r) = result(\alpha,r)$ and $out(do(\alpha/u),r) = result(\alpha/u,r)$.

Hence

$$do(\alpha)[v] = do(\alpha/u)[v] \Rightarrow result(\alpha,r) = result(\alpha/u,r)$$

and the lemma follows directly. □

Definition 1 and Lemma 6 are expressed in terms of the behaviour of the system when confronted by action sequences of arbitrary length. The following theorem establishes conditions on individual state transitions that are sufficient to guarantee security. This result is a special case of the "Unwinding Theorem" of [Gougen84].

Theorem 7: Unwinding Theorem. Let M be an internally consistent system such that $\forall u,v,w \in E,\ s,t \in V$ and $r,q \in R \mid r.\text{requestor} = u \wedge q.\text{requestor} = w$:

1) $u \not\leadsto v \Rightarrow \text{next}(s,r)[v] = s[v]$, and

2) $s[u] = t[u] \Rightarrow \text{next}(s,q)[u] = \text{next}(t,q)[u]$.

Then M is secure with respect to $\not\leadsto$.

Proof: We show that the conditions of the theorem imply

$$u \not\leadsto v \Rightarrow \text{do}(\alpha)[v] = \text{do}(\alpha/u)[v]$$

The result then follows from the previous lemma. The proof is by induction on the length of the action sequence α. The basis is the case $\alpha = \Lambda$ and is trivial. For the inductive step, suppose the theorem true whenever α is of length n and consider the action sequence $\alpha = \beta \circ x$ and $\beta \in R^*$ is an action sequence of length n. We need to prove

$$u \not\leadsto v \Rightarrow \text{do}(\beta \circ r)[v] = \text{do}(\beta \circ x/u)[v]$$

We consider two cases according to the identity of the requestor of x.

Case 1: x.requestor = u. Here $x/u = \Lambda$ and so we need

$$u \not\leadsto v \Rightarrow \text{do}(\beta \circ r)[v] = \text{do}(\beta/u)[v].$$

Now the first condition in the statement of the theorem gives

$$u \not\leadsto v \Rightarrow \text{next}(\text{do}(\beta),r)[v] = \text{do}(\beta)[v]$$

and by definition we also have $\text{do}(\beta \circ r) = \text{next}(\text{do}(\beta),r)$. Hence

$$u \not\leadsto v \Rightarrow \text{do}(\beta \circ r)[v] = \text{do}(\beta)[v].$$

The inductive hypothesis gives

$$u \not\leadsto v \Rightarrow \text{do}(\beta)[v] = \text{do}(\beta/u)[v],$$

and so we deduce

$$u \not\leadsto v \Rightarrow \text{do}(\beta \circ r)[v] = \text{do}(\beta/u)[v]$$

as required.

Case 2: x.requestor ≠ u. Here $x/u = x$ and so we need

$$u \not\leadsto v \Rightarrow \text{do}(\beta \circ x)[v] = \text{do}(\beta/u \circ x)[v].$$

Now the inductive hypothesis gives

$$u \not\leadsto v \Rightarrow \text{do}(\beta)[v] = \text{do}(\beta/u)[v]$$

and the second condition in the statement of the theorem gives

$$\text{do}(\beta)[v] = \text{do}(\beta/u)[v] \Rightarrow \text{next}(\text{do}(\beta),x) = \text{next}(\text{do}(\beta/u),x)[v]$$

whence the result follows.

□

The Unwinding Theorem is a powerful result. We now prove that unwinding is, in a certain sense, complete: for any secure system, we can find a collection of private states and abstraction functions that satisfy the conditions of Definition 5 and Theorem 7.

Theorem 8: *If M is a secure system, then for each requestor $u \in E$, a set of private states V_u and an abstraction function*

$$\text{abstract}_u : V \rightarrow V_u$$

can be found such that for all reachable states $s,t \in V$, $u,v,w \in E$ and $r,q \in R \mid r.requestor = u \wedge q.requestor = w$:

1) $s[u] = t[u] \Rightarrow out(s,r) = out(t,r)$,

2) $u \not\leadsto v \Rightarrow next(s,r)[v] = s[v]$,

3) $s[u] = t[u] \Rightarrow next(s,q)[u] = next(t,q)[u]$.

Proof: We use the following construction. First, for $u \in E$ define a relation $\sim_u$ over R^* by

$$\alpha \sim_u \beta \equiv result(\alpha \bullet \gamma, r) = result(\beta \bullet \gamma, r) \; \forall \; \gamma \in R^*.$$

It is easy to see that $\sim_u$ is an equivalence relation. Let $[\alpha]_u$ denote the equivalence class of $\sim_u$ to which the *action sequence* $\alpha \in R^*$ belongs. We define the private states of user u to be this set of equivalence classes and we define the corresponding abstraction function by

$$do(\alpha)[u] \equiv [\alpha]_u.$$

It is routine to verify that this construction is well-defined and that it satisfies the conditions in the conclusion of the Theorem. □

Theorem 8 allows us to state the following theorem.

Theorem 9: If $\not\leadsto$ is the MLS policy of definition 2, and if our model is *secure* with respect to $\not\leadsto$ then we can find an abstraction function such that for all reachable states $s,t \in V$, $u,v,w \in E$ and $r,q \in R \mid r.requestor = u \wedge q.requestor \approx w$:

1) $s[u] = t[u] \Rightarrow out(s,r) = out(t,r)$,

2) $u \not\leadsto v \Rightarrow next(s,r)[v] = s[v]$,

3) $s[u] = t[u] \Rightarrow next(s,q)[u] \approx next(t,q)[u]$.

Proof: *Define the abstraction function and show* that the rules of the model imply the conditions of the Theorem.

To capture the notion of two states looking the same to a requestor the obvious *first step is to say that everything it is permitted to see in each state is the* same

$$\begin{aligned} &\forall\, u \in E,\ s,t \in V \\ &\quad \{\, e : \mathbf{dom}\ s \mid s.Class(e) \lesssim s.Class(u) \,\} \lhd s \\ &\qquad = \{\, e : \mathbf{dom}\ t \mid t.Class(e) \lesssim t.Class(u) \,\} \lhd t \\ &\quad \mathbf{dom}\ s = \mathbf{dom}\ t \end{aligned}$$

While this is obviously necessary is it sufficient? To capture *fully the first* condition of the theorem it must also be the case that the modification options of the entity are the same in both states.

For entities below the users classification modification is not an option because of Confidentiality1. For entities at and above the users level modification is subject to the constraints of Separation_of_Duty which relates Role, Trust and Conflict controls. The set of entities at or below the users level and their Role, Trust and Conflict controls will be identical because of the observation constraint above. For entities above the users level the set of modifiable entities is controlled by Conflict which must therefore also be constrained to be identical. Trust and Role, although involved in Separation of Duty, concern only the requestors, which by Confidentiality1 and the constraints of W that requestors are members of both observers and observed, ensures that requestors must be at the level of the user. Thus Trust and Role of higher fellow requestors cannot be probed and no additional contraints are required on Trust or Role. Thus we require in addition the following constraint:

$$\forall u \in E, s,t \in V$$
$$\mathrm{dom}(s.Conflict \rhd \{s.Role(u)\}) \lhd s.Conflict = \mathrm{dom}(t.Conflict \rhd \{t.Role(u)\}) \lhd t.Conflict$$

The Class of higher entities cannot be probed in this way because Confidentiality overrides all mechanisms which a requestor could use. Thus the degree to which another entity is higher, or, whether it is even classified, is invisible to the requestor.

Thus our abstraction function, that an entities view of states is identical (EVI) is as follows:

EVI

$u : E$

$s,t : V$

$$\mathbf{dom}\ s = \mathbf{dom}\ t \qquad (a)$$

$$\{ e : \mathbf{dom}\ s \mid s.Class(e) \leq s.Class(u) \} \lhd s = \{ e : \mathbf{dom}\ t \mid t.Class(e) \leq t.Class(u) \} \lhd t \qquad (b)$$

$$\mathrm{dom}(s.Conflict \rhd \{s.Role(u)\}) \lhd s.Conflict = \mathrm{dom}(t.Conflict \rhd \{t.Role(u)\}) \lhd t.Conflict \qquad (c)$$

The proof of this is quite tedious and involes showing that for each of the classes of transitions (rules) which we have defined for the model the above abstraction function implies each of the three aspects of the unwound policy

1) $s[u] = t[u] \Rightarrow out(s,r) = out(t,r)$,

2) $u \not\leftrightarrow v \Rightarrow next(s,r)[v] = s[v]$,

3) $s[u] = t[u] \Rightarrow next(s,q)[u] = next(t,q)[u]$.

The rules have conditions in them designed to fulfill the axioms and it is the fact that unchanged they also uphold this orthoganal view which provides us with confidence in their correctness.

The Change Attribute Rule

ChangeAttribute

W

```
d! = "yes"                                                              (1)
v' ≠ v                                                                  (2)
dom v' = dom v                                                          (3)
(observers ∪ r?.observed) ∩ dom v ⊆ dom v.Class
(observers ∪ r?.observed) ∩ dom v' ⊆ dom v'.Class                       (4)
GLB v.Class( modified ) ≥ LUB v.Class( r?.observed )                    (5)
  (v.Class ↑ v'.Class ≠ ∅) ∨
  (v.Conflict ↑ v'.Conflict ≠ ∅)                                        (6)
⇒
  ( ∀ r : r?.requestors • dont_signal ∈ v.Trust({r}) )

  ∀ e : modified_controls •
      v.Role( faithful_requestors ) ⊇ v.Conflict( {e} )                 (7)
where
  faithful_requestors ≙ { r : r?.requestors |
                          (r↦faithful) ∈ v.Trust    }                   (8)
```

The First Unwound Condition

Strategy is to show that all the preconditions of rule which affect the output are implied to be identical by EVI

State dependent conditions of ChangeAttribute which might affect output of ChangeAttribute are:

4) (observers ∪ r?.observed) ∩ dom v ⊆ dom v.Class
5) GLB v.Class(modified) ≥ LUB v.Class(r?.observed)

EVI does not distinguish between entities which are not classified and those which are classified higher when defining the private view of u (b). As the conditions which discriminate these two aspects of Class are conjoined in ChangeAttribute the rule as a whole cannot distinguish differences in the whole Class relation which are not distinguished by EVI.

6) v.Trust(requestors) and dont_signal

ChangeAttribute requires that all requestors which cooperate in a transition which modifies an entities class or conflict must have the dont_signal trust attribute(6). Because of (5) and the definition of W all requestors are the same level as u whereby (b) of EVI requires that Trust, amongst others, is equal for such entities.

7) v.Role(requestors) ⊇ v.Conflict({e})
8) v.Trust(requestors) and faithful

ChangeAttribute requires that all requestors which cooperate to modify an entities controls posses faithful in Trust (8) and their combined roles through Role are a superset of roles through Conflict of the modified entity (7). Because of (5) all requestors are at the same level as u whence via (b) Role and Trust are the same for all requestors where this would matter. The Conflicts of all entities which u can modify by virtue of (5) (ie where it matters) are the same via (c) and those which it can't modify because of (5) are the same via (b).

Thus ChangeAttribute supports the first condition of the unwound policy.

The Second Unwound Condition

Strategy is to show that all modifications that can be made by ChangeAttribute resulting from requestors including a higher requestor are not visible via EVI to requestors involving a lower requestor.

This proof will fail for transitions which invoke the dont_signal trust attribute, and are thus defined to not be "real" failures of the model.

The definition of u $\not\leadsto$ v is ¬(v.Class(u) ≤ v.Class(v)).

We consider each case of EVI in turn.

Case a) Requires that **dom** s = **dom** t.

This is trivially satisfied by (3) from the definition of ChangeAttribute.

Case b) Requires that { e : **dom** s | s.Class(e) ≤ s.Class(v) } ◁ s = { e : **dom** t | t.Class(e) ≤ t.Class(v) } ◁ t

For this we need a little lemma.

Lemma:

$$\forall x,y,z : CLASS \bullet \neg(x \leq y) \wedge (x \leq z) \Rightarrow \neg(z \leq y)$$

Proof: ("For those who have the mathematical ability of a concussed bee".PJM)

By definition of ⇒
$$\neg(\neg(x \leq y) \wedge (x \leq z)) \vee \neg(z \leq y)$$

By DeMorgan
$$\neg(\neg(x \leq y) \wedge (x \leq z) \wedge (z \leq y))$$

By Transitivity of ≤
$$\neg(\neg(x \leq y) \wedge (x \leq y))$$

Which is identically true. □

(i) From definition of $\not\leadsto$
$$u \not\leadsto v \Rightarrow \neg(s.Class(u) \leq s.Class(v))$$

(ii) From definition of ChangeAttribute (5)
GLB s.Class⟦ modified ⟧ ≥ LUB s.Class⟦ r?.observed ⟧

(iii) From definition of W
u ∈ r?.observed

(iv) From (ii) and (iii) and definition of LUB
GLB s.Class⟦ modified ⟧ ≥ s.Class(u)

(v) From definition of GLB and (iv)
∀ m ∈ modified • s.Class(m) ≥ s.Class(u)

(vi) From (i) and (v) and lemma we have
¬(s.Class(m) ≤ s.Class(v))

(vii) From (vi)
modified ∩ { e : **dom** s | s.Class(e) ≤ s.Class(v) } = {}

(viii) If t.Class = s.Class
modified ∩ { e : **dom** t | t.Class(e) ≤ t.Class(v) } = {}

(ix) From definition of modified in W and (vii) and (viii)
follows (b)

(x) From (6)
s.Class ≠ t.Class ⟶ ∀ r : r?.requestors • dont_signal ∈ v.Trust⟦{r}⟧

Thus ChangeAttribute supports (b) within the limits we have defined.

Case c) Requires that dom(s.Conflict ▷ {s.Role(v)}) ◁ s.Conflict
= dom(t.Conflict ▷ {t.Role(v)}) ◁ t.Conflict.

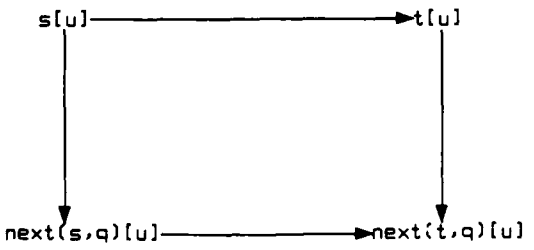
This is trivially supported by (6) from the definition of ChangeAttribute which ensures that if Conflict is modified then requestors possess dont_signal.

Thus ChangeAttribute supports (c) within the limits we have defined.

Thus ChangeAttribute supports the second condition of the unwound policy within the limits we have defined.

The Third Unwound Condition

If w and u are such that ¬(s.Class(w) ≤ s.Class(u)) then proof is same as for the second condition applied in turn to showing that s[u] = next(s,q)[u] and t[u] = next(t,q)[u] thus providing a commutative square.



If w and u are such that s.Class(w) ≤ s.Class(u) then next(s,q)[u] ≠ next(t,q)[u] implies there are transitions permitted/not permitted to w in s which are not so permitted/not permitted in t. This is equivalent to violating the first condition of the unwound theorem.

Thus ChangeAttribute supports the third condition of the unwound theorem.

Thus ChangeAttribute supports the Unwound Theorem under EVI.

The Entity Gain Rule

EntityGain

W

d! = "yes"	(1)
v' ⊐ v	(2)
dom (v' − v) ∩ **dom** v = ∅	(3)
rng v' = **rng** v	(4)
(observers ∪ r?.observed) ∩ **dom** v ⊆ dom v.Class (observers ∪ r?.observed) ∩ **dom** v' ⊆ dom v'.Class	(5)
GLB v.Class(modified) ≥ LUB v.Class(r?.observed)	(6)
∀ r : r?.requestors • dont_signal ∈ v.Trust({r})	(7)
dom v'\ **dom** v ≠ ∅ ⟶ creator ∈ v.Trust(r?.requestors)	(8)

The First Unwound Condition

Strategy is to show that all the preconditions of rule which affect the output are implied to be identical by EVI

State dependent conditions of EntityGain which might affect output of EntitGain are:

5) (observers ∪ r?.observed) ∩ **dom** v ⊆ dom v.Class
6) GLB v.Class(modified) ≥ LUB v.Class(r?.observed)

EVI does not distinguish between entities which are not classified and those which are classified higher when defining the private view of u (b). As the conditions which discriminate these two aspects of Class are conjoined in EntityGain the rule as a whole cannot distinguish differences in the whole Class relation which are not distinguished by EVI.

7) dont_signal ∈ v.Trust(requestors)

EntityGain requires that all requestors which cooperate in the transition must have the dont_signal trust attribute. Because of (6) and the definition of W all requestors are the same level as u whereby (b) of EVI requires that Trust, amongst others, is equal for such entities.

8) creator ∈ v.Trust(r?.requestors)

Similarly, EntityGain requires that at least one requestor has the appropriate trust, creator, for assigning Class, Trust, Role and Conflict attributes to the new entities. Because of (6) and the definition of W all requestors are the same level as u whereby (b) of EVI requires that Trust, amongst others, is equal for such entities.

Thus EntityGain supports the first condition of the unwound theorem.

The Second Unwound Condition

Strategy is to show that all modifications that can be made by EntityGain resulting from requestors including a higher requestor are not visible via EVI to requestors involving a lower requestor.

We consider each case of EVI in turn.

Case a) Requires that **dom** s = **dom** t.

From (3) in the definition of EntityGain we can see that this fails but as (7) requires all requestors possess dont_signal EntityGain supports a within the limitations we have defined.

Case b) Requires that { e : **dom** s | s.Class(e) ≤ s.Class(v) } ◁ s
= { e : **dom** t | t.Class(e) ≤ t.Class(v) } ◁ t.

The proof for this follows that of ChangeAttribute upto step x). At this point for EntityGain we appeal to 7) of EntityGain that all requestors are unconditionally trusted not to signal. However, the original motivation for this constraint was the Confidentiality 2 axiom that entities had been created. Thus we should also appeal to the fact that for this rule if Class is modified (in practice a certainty) it will only be for new entities, (3), and the requestors are trusted to add to Class by virtue of (8). For our defined limitations of the non-interference proof it is the former reason which formally saves us. This shift in emphasis should be noted when carrying out the real world interpretation of dont_signal and new_class.

Thus EntityGain supports (b) within the limits we have defined.

Case c) Requires that dom(s.Conflict ▷ {s.Role(v)}) ◁ s.Conflict
= dom(t.Conflict ▷ {t.Role(v)}) ◁ t.Conflict

This is trivially supported by (7) which ensures that the requestors unconditionally possess dont_signal. As above however we note that this involves a shift of emphasis from the original motivation where (3) and (11) ensured that only "additions" to Conflict for the newly created entities are made in a trustworthy manner.

Thus EntityGain supports (c) within the limits we have defined.

Thus EntityGain supports the second condition of the unwound policy within the limits we have defined.

The Third Unwound Condition

The proof of this proceeds as for Change Attribute.

Thus EntityGain supports the third condition of the unwound theorem.

Thus EntityGain supports the unwound theorem under EVI.

The Entity Loss Rule

EntityLoss

W

$$d! = \text{"yes"} \quad (1)$$
$$v' \sqsubset v \quad (2)$$
$$\mathbf{dom}(v - v') \cap \mathbf{dom}\ v' = \emptyset \quad (3)$$
$$\mathbf{rng}\ v' = \mathbf{rng}\ v \quad (4)$$
$$(observers \cup r?.observed) \cap \mathbf{dom}\ v \subseteq dom\ v.Class$$
$$(observers \cup r?.observed) \cap \mathbf{dom}\ v' \subseteq dom\ v'.Class \quad (5)$$
$$GLB\ v.Class(\!|\ modified\ |\!) \geq LUB\ v.Class(\!|\ r?.observed\ |\!) \quad (6)$$
$$\forall\ r : r?.requestors \bullet dont\_signal \in v.Trust(\!|\{r\}|\!) \quad (7)$$
$$\forall\ e : modified\_controls \bullet v.Role(\!|\ faithful\_requestors\ |\!) \supseteq v.Conflict(\!|\ \{e\}\ |\!) \quad (8)$$

where

$$faithful\_requestors \triangleq \{\ r : r?.requestors \mid (r \rightsquigarrow faithful) \in v.Trust\ \} \quad (9)$$

The First Unwound Condition

Strategy is to show that all the preconditions of rule which affect the output are implied to be identical by EVI

State dependent conditions of EntityGain which might affect output of EntityLoss are:

5) $(observers \cup r?.observed) \cap \mathbf{dom}\ v \subseteq dom\ v.Class$
6) $GLB\ v.Class(\!|\ modified\ |\!) \geq LUB\ v.Class(\!|\ r?.observed\ |\!)$

EVI does not distinguish between entities which are not classified and those which are classified higher when defining the private view of u (b). As the conditions which discriminate these two aspects of Class are conjoined in EntityLoss the rule as a whole cannot distinguish differences in the whole Class relation which are not distinguished by EVI.

7) $dont\_signal \in v.Trust(\!|\ requestors\ |\!)$

EntityLoss requires that all requestors which cooperate in the transition must have the dont_signal trust attribute. Because of (6) and the definition of W all requestors are the same level as u whereby (b) of EVI requires that Trust, amongst others, is equal for such entities.

8) v.Role(requestors) ⊇ v.Conflict({e})
9) v.Trust(requestors) and faithful

EntityLoss requires that all requestors which cooperate to modify an entities controls posses faithful in Trust (9) and their combined roles through Role are a superset of roles through Conflict of the modified entity (8). Because of (6) all requestors are at the same level as u whence via (b) Role and Trust are the same for all requestors where this would matter. The Conflicts of all entities which u can modify by virtue of (6) (ie where it matters) are the same via (c) and those which it can't modify because of (6) are the same via (b).

Thus EntityLoss supports the first condition of the unwound theorem.

The Second Unwound Condition

Strategy is to show that all modifications that can be made by EntityLoss resulting from requestors including a higher requestor are not visible via EVI to requestors involving a lower requestor.

We consider each case of EVI in turn.

Case a) Requires that **dom** s = **dom** t.

From (3) in the definition of EntityLoss we can see that this fails but as (7) requires all requestors possess dont_signal EntityGain supports a) within the limitations we have defined.

Case b) Requires that { e : **dom** s | s.Class(e) ≤ s.Class(v) } ◁ s
= { e : **dom** t | t.Class(e) ≤ t.Class(v) } ◁ t.

The proof for this follows that of ChangeAttribute upto step x). At this point for EntityLoss we appeal to 7) of EntityLoss that all requestors are unconditionally trusted not to signal.

Thus EntityGain supports (b) within the limits we have defined.

Case c) Requires that dom(s.Conflict ▷ {s.Role(v)}) ◁ s.Conflict
= dom(t.Conflict ▷ {t.Role(v)}) ◁ t.Conflict

This is trivially supported by (7) which ensures that the requestors unconditionally possess dont_signal.

Thus EntityLoss supports (c) within the limits we have defined.

Thus EntityLoss supports the second condition of the unwound policy within the limits we have defined.

The Third Unwound Condition

The proof of this proceeds as for ChangeAttribute.

Thus EntityLoss supports the third condition of the unwound theorem.

Thus EntityLoss supports the unwound theorem under EVI.

Thus EntityGain, EntityLoss and ChangeAttribute support the unwound theorem under the abstraction function EVI. Hence the rules of our model support Theorem 9 and our model is therefore Multi-Level Secure. □

The Safe Entity Gain Rule

This approach also allows us to go beyond what we can do with the axioms, an example of [Guttman87]. We can formulate a second variant of the EntityGain rule whereby instead of appealing to the trusted status of the requestors we constrain the rule to create "safe" controls on the new entities. This rule forms then the basis of a transition which can be used by "untrusted" requestors.

SafeEntityGain

W

$$v' \supseteq v \quad (1)$$

$$\mathbf{dom}(v' - v) \cap \mathbf{dom}\ v = \emptyset \quad (2)$$

$$\mathbf{rng}\ v' = \mathbf{rng}\ v \quad (3)$$

$$(observers \cup r?.observed) \cap \mathbf{dom}\ v \subseteq dom\ v.Class$$
$$(observers \cup r?.observed) \cap \mathbf{dom}\ v' \subseteq dom\ v'.Class \quad (4)$$

$$GLB\ v'.Class(\ observers\ ) \geq LUB\ v.Class(\ r?.observed\ ) \quad (5)$$

$$\forall\ r : r?.requestors \bullet dont\_signal \in v.Trust(\ \{r\}\ ) \quad (6)$$

$$v'.Trust = v.Trust \quad (7)$$

$$\forall\ o : E \mid o \in \mathbf{dom}(v' - v) \bullet$$
$$v'.Role\ (\ o\ ) \in v.Role\ (\ r?.requestors\ ) \quad (8)$$
$$GLB\ v'.Class\ (\ v'.Role^{-1}\ (v'.Conflict\ (\ o\ ))) \geq LUB\ v.Class\ (\ r?.observed\ ) \quad (9)$$
$$\#\ v'.Conflict(\ \{o\}\ ) \geq 2$$

As the rule includes a covering constraint of Confidentiality1, (5)(1)&(2), it supports this axiom.

As the preconditions of the rule imply the first disjunct of Confidentiality2 the modified predicate is included in the rule thereby trivially satisfying Confidentiality2.

The preconditions of this rule, (1)&(2), mean that no entities are modified therefore separation of duty is trivially true.

The preconditions of the rule imply that Trusted_Creation applies. As the rule doesn't include the predicates of that axiom it obviously does not support that axiom. Instead, this rule mandates a safe form for each control.

It is in establishing that these well intentioned, but formally arbitrary, constraints are sufficient that the non-interference proof comes to the fore.

The First Unwound Condition

Strategy is to show that all the preconditions of rule which affect the output are implied to be identical by EVI

State dependent conditions of SafeEntityGain which might affect output of SafeEntityGain are:

4) (observers ∪ r?.observed) ∩ **dom** v ⊆ dom v.Class

5*) GLB v.Class(modified) ≥ LUB v.Class(r?.observed)

5* is derived from 1, 2, 5, & W which together imply it.

EVI does not distinguish between entities which are not classified and those which are classified higher when defining the private view of u (b). As the conditions which discriminate these two aspects of Class are conjoined in SafeEntityGain the rule as a whole cannot distinguish differences in the whole Class relation which are not distinguished by EVI.

6) v.Trust(requestors) and dont_signal

SafeEntityGain requires that all requestors which cooperate in the transition must have the dont_signal trust attribute(6). Because of (5) and the definition of W all requestors are the same level as u whereby (b) of EVI requires that Trust, amongst others, is equal for such entities.

Thus SafeEntityGain supports the first condition of the unwound theorem.

The Second Unwound Condition

Strategy is to show that all modifications that can be made by SafeEntityGain resulting from requestors including a higher requestor are not visible via EVI to requestors involving a lower requestor.

We consider each case of EVI in turn.

Case a) Requires that **dom** s = **dom** t.

This is supported only inasmuch that (6) means that the failure is within the limits which we have defined.

Case b) Requires that { e : **dom** s | s.Class(e) ≤ s.Class(v) } ◁ s = { e : **dom** t | t.Class(e) ≤ t.Class(v) } ◁ t

This proceeds as normal except that 5* is substituted for the usual GLB v.Class of modified whence via the definition of observers as a superset of modified and the fact that 1 and 2 ensure that v'.Class of modified is equal to v.Class of modified the proof still works.

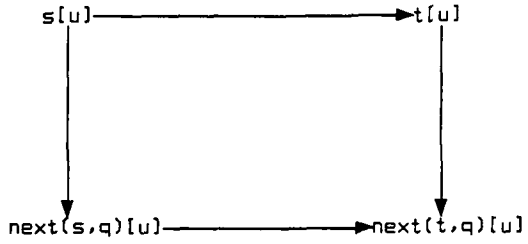
Case c) Requires that dom(s.Conflict ▷ {s.Role(v)}) ◁ s.Conflict = dom(t.Conflict ▷ {t.Role(v)}) ◁ t.Conflict.

This is satisfied by (9) which clearly is not the sort of rule condition which one would intuitively leap too when building a safe axiom busting rule. Note that this variant of the rule could be used in ChangeAttribute instead of appealing to dont_signal of Confidentiality 2.

Thus SafeEntityGain supports the second condition of the unwound theorem.

The Third Unwound Condition

If w and u are such that ¬(s.Class(w) ≤ s.Class(u)) then proof is same as for the second condition applied in turn to showing that s[u] = next(s,q)[u] and t[u] = next(t,q)[u] thus providing a commutative square.



If w and u are such that s.Class(w) ≤ s.Class(u) then next(s,q)[u] ≠ next(t,q)[u] implies there are transitions permitted/not permitted to w in s which are not so permitted/not permitted in t. This is equivalent to violating the first condition of the unwound theorem.

Thus SafeEntityGain supports the third condition of the unwound theorem.

Thus SafeEntityGain supports the unwound theorem under EVI.

Thus our model augmented with the SafeEntityGain rule also supports the unwound theorem under EVI, supports therefore Theorem 9 and is therefore also Multi-Level Secure.

4. THE SMITE ARCHITECTURE

This section examines the fundamental mechanisms required to implement the Security Policy Model. It then introduces the basic architecture of the SMITE multi-processor by first describing the capability addressing nature of the architecture, followed by the basic protection mechanisms used to provide the discrete components of the policy model abstraction.

The Basic Requirements of the Model

The basic elements of the policy model abstraction are Entities, Attributes, their Relationships, and the state transition Rules.

In the formal model Entities and Attributes are inscrutable elements of the sets E and A. There is no formal sense in which an Entity or Attribute is, or contains, data, or anything else. The only notion of change in the model are the state Relations.

We are interpreting the Relations as representing the notion of "access" to Attributes by an Entity. Our interpretation of the model has no concern for what a notion of "access" means except that the semantics of access never changes in the face of spatial or temporal multiple accesses. In real terms this implies that the only important notion of access which we wish to capture is that of being able to "name", "address" or "reference". Thus Entities cannot conceive of accessing Attributes which are not mapped from them by a state Relation. The model assumes that all Entites can "name", "address", "reference" all other Entities.

The model representation of a Request is that the "requesting entities" present a subset of the Entities "which they can name", "to which they have access". It is the transition Rule which observes, changes and modifies the state Relations in the light of the set of the offered and/or requesting Entities and the Attributes associated with them by the state Relations.

Viewed in this way Entities and Attributes are simply immutable things which can be named and used to index in and out of centrally stored state Relations which are visible only to the transition Rules.

An alternative interpretation is to view Entities as "containing" the names, or references, to the Attributes in their component of a state Relation. In other words the state Relations are the sum of the contents of Entities. This view is dependent on a number of assumptions in order to provide an interpretation consistent with the model.

Firstly, Attributes must have protected contents, such as a read-only memory segment or disc file containing textual data etc, because the model assumes that Attributes are immutable and convey no information by means of a change of value during or between state changes.

Secondly, the addressability of Attributes is strictly controlled in that only entities which have obtained the attribute address in a valid state transition can address an attribute. Thus an entity must not be able to forge the address of arbitrary attributes.

Entities are represented by objects whose contents, references to Attributes, can be protected from observation and modification by other Entities but not by transition Rules. Note, that it is the contents of Entities which must be protected, the model makes no assumptions about restrictions on the addressability of Entities. In practice, having implemented the protection of both contents and address for attributes the same mechanism will be used for entities. While this implementation restriction appears useful, no use of it is made in the model.

These requirements can be simply interpreted as the requirement for a two state machine and the model is therefore a generic security model which can be implemented with any conventional Trusted Computer Base technology.

The architecture proposed for implementing the model in the SMITE project is a capability addressing processor. Experienced readers may wonder why, having shown that a conventional TCB approach is necessary and sufficient for the models implementation, we are proposing a capability processor. Capability architectures have a bad reputation for implementing secure systems despite the intuitively attractive features which appear to make them ideal for this purpose: fine grain, flexible, hardware enforced protection. We believe that this reputation is undeserved.

Our conjecture is that capability machines have failed in the past because of the inappropriateness of the security policy models which they have attempted to implement. In other words we regard the limiting factor as the models that they have implemented rather than the basic architecture. We use the capability protection to provide the attribute address and contents protection. This is also used for protecting the contents of entities with the result that entity addressing is also constrained by the capability implementation in a manner not required by the model. It is in attempting to model a fine grained capability architecture with policy enforcement at this level of access to entities which has driven past efforts into failure because of the complexity of capability distribution and revocation control. Our policy is not enforced at this level and is we believe the reason why SMITE will succeed where past efforts have failed.

In pursuing this argument it is important that we avoid confusion between the SMITE meaning of terms, such as capability protection, and the complex concepts which a reader with some knowledge of previous capability architectures may possess. In order to achieve this we will therefore describe the very simple notions that we use in SMITE before showing its correspondence to the model.

The SMITE Basic Architecture

Capability addressing is simply the concept that areas of memory are addressed by a pointer. The difference between a capability and say a conventional base or index addressing mode is that a word representing a capability is fundamentally distinguished from a word representing a scalar number. Thus in a conventional architecture a word representing a number can be used in arithmetic instructions by loading it into "accumulator" registers or it may be used in address calculation of memory store and fetch instructions by loading it into a "base" or "index" register.

In a capability architecture the registers and words in store used to address store are typed differently from normal scalar registers and words and the different roles are enforced by the various instructions of the instruction set. This "typing" of words may be achieved by additonal "hidden" tag bits or by segregation of the store, this being an implementation detail. Capabilities also enforce the bounds of the area of memory addressed, by a size or range field within the capability, and thus represent a stronger addressing mode than the conventional base or index register alone where the memory to be addressed is limited solely by the size of the offset register.

The use of capabilities, of a base-range form, as the only mode of addressing and the fundamental enforcement of the distinction between capability and scalar data is the only notion of protection which SMITE implies by the use of capability protection. A capability cannot be forged or guessed by manipulation of scalar data, a scalar word with the same bit pattern as a capability word are not the same, by virtue of the address of the word or by the hidden tag bit in a segregated or tagged architecture respectively.

There are no instructions to "generate" a capability per se in a capability machine, instead instructions which generate new objects generate new unique capabilities to address them. Capabilities can be freely copied to enable sharing of objects.

Thus far SMITE and previous capability approaches share common concepts. The differences between SMITE and other capability architectures arises from the degree and method of use of the additional mechanisms which are built onto the basic capability pointer and protection idea. A generic development of these mechanisms and the contrasting uses of SMITE and conventional architectures is developed below simply to provide the necessary volcabulary for discussing the SMITE implementation of mechanisms corresponding to constraints of the policy model.

Words used as capabilities are essentially private data types of the machine instruction set used in address calculation. Because of this private nature capabilities and their instructions can be further "typed" by using "spare" bits of the word, ie those bits not required to form a pointer. Thus bits have been used to provide read, write, and execute access control on the store addressed by the capability, bits have been used to control entry and exit from "supervisor" or "privilege" states of code accessed by execute access capabilities, and bits have been used to provide extensive typing to produce high-level object based language processors [Tyner81]. Past efforts to turn the access control nature of read write access capabilities into full blown policy enforing architectures, with security labels and built in "dominates" rules in the instruction set, have also been attempted.

SMITE uses the "extra" bits to define a number of different types of capabilities. SMITE uses the term "block" and "block type" to denote this on the basis that in practice a capability defines some block of store in terms of a base and offset. Thus the instruction set is partitioned into instructions which work on different capability types where the type defines the format and interpretation of the contents of a block [Harrold88b, Cooper87].

SMITE further defines a single bit for a capability status, "Locked" or "Unlocked". For each capability type the influence of the lock bit is defined as a modification on the permitted modes and areas of access to the block contents. The exact interpretation is dependent on the block/capability type. For most types unlocked denotes unconstrained access to the entire block contents. For some types locked functions as a modification inhibit bit, for other types it may act as a total bar on access to the block contents. Again the protection status conferred in the locked/unlocked state may extend to the entire contents of the block or only to some initial number of words dependent on the block/capability type. The instruction set creates capabilities to blocks in the unlocked or open state and provides an instruction for locking a capability. (NB, lock is a capability status bit not a block status). There is not an instruction for unlocking a capability.

Given this much condensed description of the basic SMITE architecture we can *now begin to describe the interpretation of the model*. Rather than present this en masse we will develop the requirements in a tutorial fashion and subsequently summarise the interpretation in terms of the full SMITE architecture. The tutorial therefore presents the two block types from which all of the basic protection mechanisms are built. For full understanding of the supporting types the reader should refer to the instruction set specification [Harrold88b].

The Basic Protection Mechanisms

In a capability architecture Entities and Attributes are capability addressed blocks. The Relationship between Entities and Attributes is represented by the capabilities of Attributes stored in Entity blocks. Only a subset of the capability addresses are equated to Entities and Attributes which may thus posses capabilities to other blocks which have no corresponding element in the policy model. These blocks can therefore be regarded as the primitive building blocks from which Entities and Attributes are built. The policy model and its axioms are built on the notion that Entities and Attributes are discrete, independent items related only in the ways explicitly defined by the model. There is thus an obligation to show a "proof of independence" on the capabilities used to build Entities and Attributes.

The instruction set provides mechanims to manipulate capabilities to blocks and the contents of blocks in a primitive manner which cannot be cognisant of the policy model versus primitive permitted operations. We therefore require "blocking" or "hiding" mechanisms to protect the policy model mappings from arbitrary instruction set manipulation and yet to allow the application of policy model rule transitions, which themselves can only be sequential applications of individual instructions of the primitive instruction set stored in a capability addressed block.

We will develop the semantics of the required hiding mechanism in a rather tutorial fashion so as not to introduce any false connotations.

Define a block type for which the capability LOCK bit semantics are any access versus no access. If the locked capabilities to these blocks are used for representing Entities no one can alter the relationships or obtain copies of the attribute capabilities contained therein. In addition if these locked capabilities are also used to represent Attributes arbitrary modification cannot violate the "independence" and/or immutability constraints required with respect to primitive blocks.

We require that a sequence of instructions which represents a state transition rule can access the block and manipulate the block contents.

Note it is the state transition rule which must access the block not the instigating active entity, the requestor in model terms. Thus we require that the transition rules are represented in some way which combines not only the constraints, ie the particular sequence of instructions, but protected access to entities, ie only they posses the open capabilities. This representation must exist as a capability addressed block within the system and thus the open capability within it must be hidden from arbitrary access which could steal it and carry out unconstrained transitions on the model relationships.

Thus the transition rule representation must have capabilities whose semantics are that only invocation, or execution, is allowed. These blocks in SMITE are called closures.

A closure block contains two words, a capability to a block containing scalar and capability words, and a capability to a block containing scalars which are interpreted as instructions.

The principle property of the closure type is that closures can only be invoked and that the two pointers are hidden and inaccessible given only a capability to the closure block. Conversely, the only view of the system available to a closure when invoked is its own scalar/capability block and the parameters supplied by the caller.

In the closure regime sensitive values, such as the open capabilities to entities and attributes, required by code, such as the transition rules, are stored in the scalar/capability block. This is used by the transition rule code when invoked and is inaccessible to the caller, before, during and after the call. The code of a closure, being software, is almost infinitely variable in the exercise of checks that can be made upon the parameters of the call in deciding whether to proceed with the access to the sensitive data structure.

The closure scalar/capability block and code block form a tailored protection environment upon each call and thus serve in the same way as protection rings, supervisor states etc of two state machines except that they are not fixed in either form, function or number by the system but exist in a flexible, distributed form. This approach was correctly identified in the Plessey 250 capability machine [England75], where the enter capability was used to structure system software in the same way. Sadly this effort failed primarily for non-technical reasons though as we shall see below there are some other aspects of the SMITE system which the System 250 lacked.

An implementation conformant with the policy model could be mounted with these as the basic protection mechanims, a black-box block and closure containing the open capabilities to the contents of the black-boxes. The system would probably have to be fairly static in terms of the creation and destruction of entities and attributes because the system structuring requirements to instantiate closures containing capabilities to the new objects without security compromise can become onerous with only closure and black-box protection.

The Plessey 250 carried out this instantiation at link time for the system build using a cumbersome link language. In SMITE the interpretation of execution access protection is fully supportive of Landin's Closure notion [Landin64], which includes run time support.

The concept of the Closure based software regime is simply that the code of a procedure is bound to its environment, defined by its code constants and non-local variables, to form an independent executable unit which may be applied to parameters but more importantly can be stored and manipulated as for any other data. Thus it may be passed as parameters and returned as results and stored in data structures giving rise to a number of elegant language and software properties that give the concept another name: first class procedures, [Currie82].

This is simply implemented on SMITE with the closure scalar/capability block containing the non-locals and the code block the procedure body. The ability to mix capability and scalar data within a block is an advantage of a tagged capability architecture such as SMITE which the segregation architecture of the System 250 lacked. Absence of this feature adds an extra level of complexity when mapping a procedures non-locals, which may include scalars and capabilities, making a simple implementation of the pure software closure concept difficult.

Even with this high level of run-time support on SMITE the instantiation problem for new black-boxes becomes onerous because of problems which arise from a system containing both trusted and untrusted code. The problem is essentially that trusted and untrusted closures cannot be differentiated thus allowing spoofing during the distribution of the new closures. In order to mitigate this problem the basic black-box mechanism is extended to a notion of typed objects.

Instead of instantiating a set of transition rule representations for access to each instance of black-box we provide a single set which can accept the locked capability to a black-box as a parameter and then "open" the black box capability within the protective domain of the closure environment. This requires the provision of an unlock instruction to produce open copies of locked black-boxes. In order that we don't regress to the problem of how to stop arbitrary instruction sequences opening any black-box the unlock instruction for black-boxes is "keyed" to some unforgeable token possed by closures which are intended to access black-boxes. This is achieved by the following sematics for Keyed Blocks, the SMITE name for such selectively opened black-box capabilities.

A keyed block capability can be locked by any one but can only be unlocked by someone who can quote the first word of the contents of the block, the "key". This word cannot be forged, or guessed, if it is a capability for a block which is known only to the intended closures. For the purposes of the model, this could be a system wide unique key possed by all state transition closures and used to lock all entities and attributes. For integrity policies, the transition rules will naturally partition into groups of rules concerned with partitioned subsets of entities and attributes. This is normally expressed in the model by means of attributes for distinguishing such "types". It is relatively obvious therefore to use the key itself as the representation of such type attributes in implementing the model. This provides the damamge limitation notion of least privilege in the implementation of the transition rules and emphasises the role of keyed blocks as "typed objects".

The instantiation of the transition rules is carried out by a closure, the "type manager", which generates a block to get a "key" and then delivers the transition rule closures, the "type operators", with this key as a non local. Provided that the type manager and operations never deliver the "key" capability, or, open pointers to blocks containing the "key", as a result, the contents of all blocks accessed by the locked capabilities are immutable and hidden to all but the transition rules.

This is then the sum total of the SMITE architecture requirements for implementing the model with fine grained, flexible, hardware enforced protection: capabilit· addressing, Open/Lock capability status, Closures, Typed Objects.

Implementing the Model Elements

The implementation of each of the policy model elements is discussed below in terms of the essential behaviour requirements implied by the policy model, in other words the constraints of the refinement proof chain.

ATTRIBUTES

The essential nature of attributes is that they are immutable, tranquil objects. Once created there must be no primitive sense in which they can be modified. There are no constraints other than this on the complexity of attributes in terms of the primitive objects from which they are built.

The tranquility requirement is required so that when transition rules incorporate attributes into entities as relationships there is no variety that can be imparted to subsequent state transitions involving the attribute. For this reason state transitions require an easy and reliable method to identify primitive structures as valid candidates for use as attributes.

Implementing attributes as a read only primitive capabilities is not sufficient to ensure tranquility because a read only capability to the root of a capability structure does not imply that all elements within the structure are read only. Further more there is no way to know that the subject has not retained write capabilities to the object or its elements.

Thus we use trusted closures which create attributes by copying a primitive capability structure into a read only copy within a type protected object. The copy is unique, because only the creating closure can possibly have write capabilities to it, and tranquil, because this closure will not use those capabilities to alter the attribute and will deliver only the locked capability to the containing typed object. It is thus labelled as tranquil which can quickly be identified by state transition rules. As for the basic protection, the use of many keys for typing attributes instead of a single tranquility key and additional type information is an optional optimisation and least privilege mechanism.

The untrusted code of active entities can obtain a read only copy of the primitive structure inside an attribute, for use in "algorithmic manipulation" outside of the model, by means of complimentatry trusted closures which deliver another copy. If the attribute creating closure stores "read only" capablities to the primitive copy in the attribute the second copy by the complementary closure can be optimised to delivery of the read only capability.

PASSIVE ENTITIES

In terms of its primitive structure a passive entity is little different from an attribute, a capability structure within a typed object. An entity however is exempt from the tranquility requirement in as much as the state transition rules will define permissable circumstances in which an entity may be modified.

The exact choice of the appropriate capability structure within the object such that it may fulfill the type sematics of the models state transitions but does not allow other transitions inadvertently is of critical importance to the successful implementation of a secure system.

The type sematics of an entity is implementable purely with a structure of a vector of capabilities to attributes. For state transition/entity "types" which imply a complex sematics this is obtained at the cost of complexity in the code of the transition rules in finding attributes within the entity for the particular transition required. It may therefore provide higher assurance by structuring the attribute capabilities within the entity object. Great care must then be taken however that the state transitions do not provide "visibility" of any variety in the structure over an above the sematics of the type.

ACTIVE ENTITIES

An active entity is a closure in the worksapce chain of a launched process block. Not all closures in the workspace chain of a launched process block are active entities.

Within the process block is a pointer to a workspace block. The workspace block is unique on the launching of the process and contains a pointer, in a permanently hidden area, to the closure to be run. In the open area of the workspace block are the locals and expression stack of the closure. A closure may obtain pointers to the open area of its workspace block and the non-locals block of itself and is free to use these primitive pointers in any way it sees fit.

Capabilities to Entities, Attributes and primitive data may be stored in any manner within these two areas and may be manipulated by the closure at will.

For implementing the security policy constraints as opposed to the basic model it is important that the control relations and attributes of active entities such as Trusted, Class etc cannot be modified by the untrusted code of the entity but only by invoked state transitions.

Thus within the process block is a pointer to a table called the "Process Context". This table is a vector of word pairs where the first word of a pair is considered a "name" and the second word the resolved value. If the second word is a capability arbitrarily complex values can be resolved. If the first word is a capability then the name cannot be forged. The instructions which affect the table are "store pairs" and "resolve a word given a name". Storing a value with a name that all ready exists in the table is an update.

Any closure can store a word with a new name in this table but only closures which posses the existing name can update and/or retrieve words from this table. Thus the control Attributes, which must not be "lost" or substituted by the untrusted code of the active entity, are stored in this area with capability names known only to the transition rules.

A process is launched with an empty context table and an initial closure. This initial closure must be a trusted closure which will effect either a Gain Entity state transition which will label the entity, or arrange to inherit the context of the parent process, before running the intended closure. This intended closure is the entity and can invoke other closures resulting in a new workspace and an inherited context. These new closures are thus not entities but just the code of the entity. If the closure calls a state transition which embodies an Entity Gain transition the new closure will be running as a new entity with a changed process context. When this new entity exits the process context must be restored to that of the existing entity. Thus the process context must be carried in and restored on exit from closure invocations. This is simply acheived by caching the process context table in the chain of workspace blocks of the process.

There are thus two methods of allocating new active entities, as subtasks in an existing process or as parallel tasks in a new process. A subtask or a parallel task is not necessarily a new entity. In all four cases trusted closures are required to ensure the correct labelling in terms of the model transition which is required.

Capabilities within an active subject will include, in addition to primitive blocks, those to closures, attributes and other entities which can be passed as parameters and received as results to and from closures. Closures which are not transition rules are considered primitive and are subject to the "independence proof" obligations.

TRANSITION RULES

Transition rules are closures which possess the appropriate keys for accessing entities and attributes in their non-locals and implement the policy decisions in terms of Class, Conflict, Trust etc of the calling closure (an entity) in their code structure. They are thus "trusted" programs.

The Higher Software Structure

The discussions of the SMITE processor architecture and the implementation of the model elements above have concentrated on the main store and processor aspects of the system. Obviously this is not the whole, or necessarily the most important part of the story from the top-level view of a general purpose computer system running complex C^3I and MIS applications.

This section attempts to provide the flavour of the higher level structure of software which builds on these fundamental protection mechanisms in a way which preserves the crucial security aspects while providing the necessary flexibility for the above requirements.

In practice the structure of higher level software is that of type managers described in the original frozen report before the model was developed. Notwithstanding the practical problems in the implementation of this approach, described at length in the above report, we must first show how this mechanism can be described in terms of the model.

We will consider the following procedure declarations and their intended semantic interpretation, an example taken from the frozen report.

```
Create_Text: (Vec_Char,Vec_Char) -> TextFile
Read_Text: TextFile -> (Vec_Char,Vec_Char)
Amend_Text: (TextFile,Vec_Chars)
```

These are the operators for a type TextFile which represents the structuring of vectors of characters, forming the text of a document and an identification of a user designated the author, into a TextFile and back. Characters and Vectors of Characters are primitive instruction set types. Amend represents the modification of the text portion of an existing TextFile.

```
Create_Report: (Vec_Char,TextFile) -> Report
Read_Report: Report -> (TextFile, Vec_Char)
```

These are the operators for a type representing a Report which is interpreted as a document which has been approved for release by a user representing an authority higher than the author of the document. The constructor takes a TextFile and the vector of characters which designate the approving user. The selector returns a TextFile and the user identity of the Approver as a vector of characters.

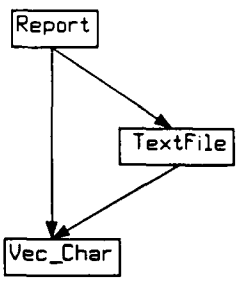
These procedures can be used in sequences representing the organisation's policy for externally releasing a document written by members of the organisation. That is, only documents which have been approved may be released.

```
report : Report
draftreport : TextFile

draftreport = Create_Text ("This is a draft report","author")

report = Create_Report("approver",draftreport)
```

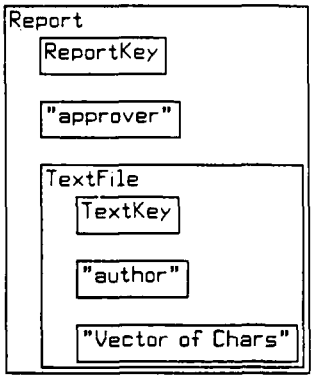
The most obvious implementation of this system is a hierarchical relationship of the types Report, TextFile, and Vec_Char as follows.



This in turn is most straightforwardly implemented in terms of keyed blocks as follows.

Instances of types Report and TextFile are keyed blocks whose key is a capability to a unique block, ReportKey and TextKey respectively. Textfile additionally contains a vector of chars of the report text and a vector of characters identifying the Author. Report additionally contains a capability to an instance of TextFile and a vector of characters identifying the approver.

This can be pictured as follows where an enclosing box represents a capability to the block named inside.



Our task is to describe this arrangement in terms of entities and attributes.

Firstly, our view of an entity as containing "names", or "references" to attributes may be felt to lead us to identify the capability as the simplest, most primitive representation of an entity. Thus

"approver"

is an entity containing the reference to the attribute

"approver"

Following this interpretation however leads us into problems. Consider interpreting TextFile above in this way. TextFile is an entity containing three entities! This is not a model concept, entities contain references to attributes not entities. Thus we cannot characterise entities and attributes in terms of simple hardware characteristics.

Instead, we can say that the encoding of an attribute resides in memory and that the address of this memory resides somewhere else in memory and that this other piece of memory is an entity. The model assumes that the memory representing the attribute cannot be arbitrarily accessed (ie its address cannot be guessed or forged) but is accessed only by transition rules on behalf of an entity which possesses the address of the entity. This requirement is simply met by SMITE in that the address of the attribute is a capability and capabilities can reside anywhere in memory. The model further assumes that entities cannot arbitrarily acquire names and references of attributes from other entities but can only recieve them by taking part in a state transition involving those entities. Once again this is simply met by applying capability protection to the area of memory representing an entity.

By this defintion TextFile is an entity containing three attributes, one which types it as a TextFile entity, one which identifies the role identifier which created this entity and the other, stuff related to the application, namely the text.

However, what is now the interpretation of Report. This is an entity which contains two attributes and the "name"/"reference" of another entity. To understand this example we must return to the model abstraction and reexamine the source of this example.

In terms of our model the application should be described as follows.

Create_Text: (Vec_Char,Vec_Char) -> TextFile

This is a state transition requiring only a single requestor, whose Role will be taken as the identity of the author, which creates a new entity, TextFile, whose Role is that of the requestor and whose Other is some subset of the requestors.

Read_Text: TextFile -> (Vec_Char,Vec_Char)

This is a state transition requiring only a single requestor, whose Role can be anything and need not be related to the authors, which returns the Role and Other of any entity whose type is TextFile.

Amend_Text: (TextFile,Vec_Chars)

This is a state transition requiring only a single requestor, whose Role must match that of the TextFile, which changes the TextFile entities Other to be some combination of the existing Other and the requestors Other.

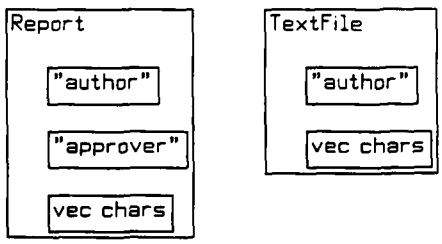
Create_Report: (Vec_Char,Vec_Char,Vec_Char) -> Report

This is a state transition requiring two requestors, an author and an approver, which creates an entity of type Report whose other is some subset of the author entities' and whose Conflict is {Role(author),Role(approver)}.

Read_Report: Report -> (Vec_Char,Vec_Char, Vec_Char)

This is state transition requiring one requestor, who can be any role, which returns the other and conflict of any entity of type Report.

In this view there is absolutely no connection or interdependence between textfiles and reports. Therefore we end up with the following straightforward "implementations".



Which is exactly what the other report arrived at after considering all of the problems if an implementation where reports are implemented with TextFile subtypes is adopted. In the implementation of reports the two requestor creation will have to be decomposed into two single user operations thus an intermediate type of the form

will be required. The resembelence of this in form to a TextFile should only encourage us to adopt that implementation strategy if all of the other implementation constraints required of proto-report are available in TextFile without compromising the functionality of TextFiles. Viewed in this light such a strategy is very unlikely and represents a vast confusion of level of abstraction and clean top down design.

Thus the original question must be unasked! An entity containing an entity should not occur in a clean application of this model.

REFERENCES

[Adams80] Adams,D.N.,"The HitchHikers Guide to the Galaxy (Fit the Ninth)", Originally broadcast by the British Broadcasting Corporation on 22 January 1980.

[Andrews81] Andrews,M.P., et al, "Multilevel Computer Security in Network Environments", Plessey Telecommunications, 1981.

[Barnes85] Barnes,D.H., "Secure Communications Processor Research", Procs. MILCOMP 85, London, October 1985.

[Bell73a] Bell,D.E. & LaPadula,L.J., "Secure Computer Systems: Mathematical Foundations", MTR-2547, Vol 1, Mitre Corp.,November 1973.

[Bell73b] Bell,D.E. & LaPadula,L.J., "Secure Computer Systems: A Mathematical Model",MTR-2547, Vol 2, Mitre Corp.,November 1973.

[Bell74] Bell,D.E., "Secure Computer Systems: A Refinement of the Mathematical Model",MTR-2547, Vol 3, Mitre Corp.,April 1974.

[Bell76] Bell,D.E. & LaPadula,L.J., "Secure Computer System: Unified Exposition and Multics Interpretation", MTR-2997,Mitre Corp.,March 1976.

[Bell88] Bell,D.E., "Concerning 'Modelling' of Computer Security", Proc. 1988 Symposium on Security and Privacy, April 1988.

[Biba77] Biba,K.J., "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, MA, April 1977.

[Clark87] Clark,D.D., & Wilson,D.R., "A Comparison of Commercial and Military Computer Security Policies", Proc. 1987 Symposium on Security and Privacy, April 1987.

[Cooper87] Cooper,D.G. & Diss,M.J., "The Specification of the SMITE Instruction Set in Z", Report Number 72/87/R528/U, Plessey Research Roke Manor Limited, December 1987.

[Currie82] Currie,I.F., "In Praise of Procedures" RSRE Memorandum No. 3499, 1982

[Currie85] Currie,I.F., Foster,J.M. & Edwards,P.W., "PERQFLEX Firmware", RSRE Report 85015, 1985.

[England75] England,D.M., "Capability Concept Mechanisms and Structure in System 250", Rev. Fr. Autom. Inf. Rech. Oper. (France), Vol 9, Sept 1975.

[Foster82a] Foster,J.M., Currie,I.F. & Edwards,P.W., " Flex: A Working Computer With An Architecture Based On Procedure Values", RSRE Memorandum 3500, 1982.

[Foster82b] Foster,J.M. & Currie,I.F. "CURT: The Command Interpreter Language for Flex", RSRE Memorandum 3522, 1982

[Foster83] Foster,J.M., Currie,I.F. & Edwards,P.W., " Kernel and System Procedures in Flex" RSRE Memorandum 3626, 1983.

[Foster86] Foster,J.M. & Currie,I.F. "Remote Capabilities", RSRE Memorandum 3947, 1986. Also Computer Journal 30 (5) October 1987, p451..457.

[Foster89] "The Algebraic Specification of a Target machine: Ten15", from "High Integrity Software", C.T.Sennett (Ed.), Pitman Press to appear 1989

[Goguen82] Goguen,J.A. & Meseguer,J., "Security Policies and Security Models", Procs. Symp. Security and Privacy, April 1982.

[Gougen84] Goguen,J.A. & Meseguer,J., "Unwinding and Inference Control", Procs. Symp. Security and Privacy, April 1984.

[Guttman87] Guttman,J., "Information Flow and Invariance", Proc. 1987 Symposium on Security and Privacy, April 1987.

[Haigh86] Haigh,J.T. & Young,W.D., "Extending the Non-Interference Version of MLS for SAT", Proc. 1986 Symposium on Security and Privacy, April 1986.

[Harrold88a] Harrold,C.L., "Formal Specification of a Secure Document Control System for SMITE",RSRE Memorandum No. 88002, 1988.

[Harrold88b] Harrold,C.L. & Wiseman,S.R., "SMITE Instruction Set Specification - Version 2", SMITE Working Paper, January 1988.

[Karger88] Karger,P., "Implementing Commercial Data Integrity with Secure Capabilities", Procs. 1988 IEEE Symp on Security and Privacy, Oakland CA., April 1988.

[Landin64] Landin,P.J., "The Mechanical Evaluation of Expressions", Computer Journal, Vol 6, Num 4,Jan 1964.

[Lee88] Lee,T.M.P.,"Using Mandatory Integrity to Enforce 'Commercial' Security", Procs. 1988 IEEE Symp on Security and Privacy Oakland CA., April 1988.

[McLean87] McLean,J., "Reasoning About Security Models", Proc. 1987 Symposium on Security and Privacy, April 1987.

[Redell74] Redell,D.D., "Naming and Protection in Extendible Operating Systems", MIT Technical Report MAC-TR-140, Nov 1974.

[Rushby81] Rushby,J.M., "Design and Verification of Secure Systems", ACM Operating System Reviews, Vol15, Num5,Dec 1981.

[Rushby85] Rushby,J.M., "Mathematical Foundations of the MLS Tool for Revised Special", Draft internal note, Computer Science Labaoratory, SRI International, Menlo Park, CA.

[Sennett87] Sennett,C.T., "Review of Type Checking and Scope Rules of the Specification Language Z", RSRE Report 87017, Novemebr 1987.

[Shockley87] Shockley,W.R.,"Implementing the Clark/Wilson Integrity Policy Using Current Technology", Gemini Computers Inc., paper submitted to the WIPCIS workshop 1987 and NBS 88.

[Spivey87] Spivey.,J.M., "The Z Notation: A Reference Manual", Draft JMS-87-12a, Programming Research Group, Oxford University, 1987.

[Terry88] Terry,P.F.& Wiseman,S.R.,"On the Design and Implementation of a Secure Computer System" RSRE Memorandum 4188, June 1988

[Terry89] Terry,P.F.,"The Smite Approach to Security" Report from RSRE Contract A94c/2711, January 1989.

[Terry-Wiseman89] Terry,P.F. & Wiseman,S.R., "A New Security Policy Model", Procs. IEEE Symp. Security and Privacy, Oakland CA, May 1989.

[Tyner81] Tyner,P., "iAPX-432 General Purpose Data Processor: Architecture ReferenceManual", Intel Corp., January 1981.

[WIPCIS87] "Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)", October 1987.

[Wiseman86a] Wiseman,S.R., "A Secure Capability Computer", Procs. IEEE Symp. Security and Privacy, Oakland CA, April 1986

[Wiseman86b] Wiseman,S.R., "A Capability Approach to Multi-level Security" Procs. IFIP/Sec, Monaco, December 1986

[Wiseman88a] Wiseman,S.R., "Protection and Security Mechanisms in the SMITE Capability Computer", RSRE Memorandum 4117, 1988.

[Wiseman88b] Wiseman,S.R. & Field-Richards,H.S., "The SMITE Computer Architecture", RSRE Memorandum 4126, 1988.

[Wiseman88c] Wiseman,S.R., Terry,P.F., Wood,A.H., & Harrold,C.L, "The Trusted Path between SMITE and the User" Procs. IEEE Symp. Security and Privacy, Oakland CA, April 1988

[Woodward87] Woodward,J.P.L., "Exploiting the Dual Nature of Sensitivity Labels",Proc. 1987 Symposium on Security and Privacy, April 1987.

DOCUMENT CONTROL SHEET

Overall security classification of sheet ....UNCLASSIFIED.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the box concerned must be marked to indicate the classification eg (R) (C) or (S))

1. DRIC Reference (if known)	2. Originator's Reference	3. Agency Reference	4. Report Security Classification
	REPORT 89014		U/C

5. Originator's Code (if known)	6. Originator (Corporate Author) Name and Location
7784000	RETIX SYSTEMS LTD 20 ALAN TURING ROAD, SURREY RESEARCH PARK, GUILDFORD GU2 5YF
5a. Sponsoring Agency's Code (if known)	6a. Sponsoring Agency (Contract Authority) Name and Location
	ROYAL SIGNALS AND RADAR ESTABLISHMENT ST ANDREWS ROAD, MALVERN, WORCESTERSHIRE WR14 3PS

7. Title

THE SMITE APPROACH TO SECURITY

7a. Title in Foreign Language (in the case of translations)

7b. Presented at (for conference papers) Title, place and date of conference

8. Author 1 Surname, initials	9(a) Author 2	9(b) Authors 3,4...	10. Date	pp. ref.
TERRY, P F			1989.8	58

11. Contract Number	12. Period	13. Project	14. Other Reference
A94c/2711			

15. Distribution statement

UNLIMITED

Descriptors (or keywords)

continue on separate piece of paper

Abstract

SMITE is a novel computer architecture implementing a new security policy model. It is proposed as the best available technology which may be used to develop information systems for operational use where high assurance of complex confidentiality and integrity based security policies is required.

This report records the results of work carried out by Retix Systems for CCI division RSRE. This contract provided a peer review of their architecture proposals, characterised the essential architecture elements and formulated the security oriented top level model. In this way it provided a baseline definition of SMITE to aid the future way forward for the project.